



CTOS

**Procedural Interface
Reference Manual
Volume 4
System Structures
And Appendixes**

UNISYS

UNISYS

CTOS[®]

Procedural Interface

Reference Manual

Volume 4

**System Structures
and Appendixes**

Copyright © 1990, 1991, 1992 Unisys Corporation
All Rights Reserved
Unisys is a trademark of Unisys Corporation

This volume has been updated for CTOS III with pages from
Update 1, 4357 4714-110

CTOS III 1.0
CTOS II 3.4
CTOS I 3.4
CTOS/XE 3.4
Development Utilities 12.2
Standard Software 12.2
Video Access Method 4.0
Priced Item

August 1992

Printed in USA

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information or software material, including direct, indirect, special or consequential damages.

You should be careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

RESTRICTED RIGHTS LEGEND. Use, reproduction, or disclosure is subject to the restrictions set forth in DFARS 252.227-7013 and FAR 52.227-14 for commercial computer software.

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association is purely coincidental and unintentional.

Convergent, Convergent Technologies, CTOS, NGEN, and SuperGen are registered trademarks of Convergent Technologies, Inc.

ClusterCard, ClusterShare, Context Manager, Context Manager/VM, CT-DBMS, CT-MAIL, CT-Net, CTOS/VM, CTOS/Vpc, Document Designer, Generic Print System, PC Emulator, Print Manager, Series 186, Series 286, Series 386, Series 286i, Series 386i, shared resource processor, SRP, TeleCluster, Voice/Data Services, Voice Processor, X-Bus, and X-Bus+ are trademarks of Convergent Technologies, Inc.

OFIS is a registered trademark of Unisys Corporation.

BTOS is a trademark of Unisys Corporation.

AT, IBM, IBM PC, and OS/2 are registered trademarks of International Business Machines Corporation. IBM PC-AT, IBM PC-XT, and IBM PS/2 are trademarks of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. MS-DOS and Microsoft are registered trademarks of Microsoft Corporation. UNIX is a registered trademark of AT&T.

Contents

About This Manual

1	Introduction to CTOS Operations	1-1
2	Operations by Function.....	2-1
3	Operations	
	AbortRequests	3-3
	AccessVersion	3-5
	AcquireByteStreamC.....	3-7
	AddQueue	3-11
	AddQueueEntry	3-13
	AllocAllMemorySL	3-17
	AllocAreaSL	3-19
	AllocateSegment	3-20A
	AllocCommDmaBuffer	3-20C
	AllocExch	3-21
	AllocHugeMemory.....	3-22A
	AllocMemoryFramesSL	3-23
	AllocMemoryInit	3-25
	AllocMemoryLL	3-27
	AllocMemorySL	3-29
	AllocUserNumbers	3-31
	AsGetVolume	3-33
	AsSetVolume	3-37
	AssignKbd.....	3-41
	AssignVidOwner	3-43
	AsyncRequest	3-45
	AsyncRequestDirect	3-47

AtFileInit	3-49
AtFileNext	3-51
Beep	3-53
BuildAsyncRequest	3-55
BuildAsyncRequestDirect	3-57
BuildFileSpec	3-59
BuildFullSpecFromPartial	3-65
BuildLdtSlot	3-69
BuildSpecFromDir	3-71
BuildSpecFromFile	3-73
BuildSpecFromNode	3-75
BuildSpecFromPassword	3-77
BuildSpecFromVol	3-81
CallRealCommIsr	3-83
CacheClose	3-85
CacheFlush	3-87
CacheGetEntry	3-91
CacheGetStatistics	3-97
CacheGetStatus	3-101
CacheInit	3-103
CacheReleaseEntry	3-107
CdAbsoluteRead	3-111
CdAudioCtl	3-115
CdControl	3-131
CdClose	3-139
CdDirectoryList	3-141
CdGetDirEntry	3-143
CdGetVolumeInfo	3-145
CdOpen	3-149
CdRead	3-153
CdSearchClose	3-155
CdSearchFirst	3-157
CdSearchNext	3-161
CdServiceControl	3-165
CdVerifyPath	3-169
CdVersion	3-171
CfaFfVersion	3-175
CfaServerVersion	3-177

CfaWaVersion	3-179
Chain	3-181
ChangeCommLineBaudRate	3-185
ChangeFileLength	3-189
ChangeOpenMode	3-191
ChangePriority	3-193
ChangeProcessPriority	3-195
ChangeUserNumberRequests	3-197
Check	3-199
CheckContextStack	3-201
CheckErc	3-203
CheckForOperatorRestartC	3-205
CheckpointBs	3-207
CheckpointBsAsyncC	3-209
CheckpointBsC	3-211
CheckpointRsFile	3-213
CheckpointSysIn	3-215
CheckReadAsync	3-217
CheckWriteAsync	3-219
CleanQueue	3-221
ClearPath	3-223
CloseAllFiles	3-225
CloseAllFilesLL	3-227
CloseAltMsgFile	3-229
CloseByteStream	3-231
CloseDaFile	3-233
CloseErcFile	3-235
CloseFile	3-237
CloseMsgFile	3-239
ClosePSSession	3-241
CloseRsFile	3-243
CloseRtClock	3-245
CloseServerMsgFile	3-247
CloseTerminal	3-249
CloseVidFilter	3-251
CloseSysCmds	3-253
CompactDateTime	3-255
ConfigCloseFile	3-257

ConfigGetNextToken	3-259
ConfigGetPosition	3-263
ConfigGetRestOfLine	3-265
ConfigOpenFile	3-267
ConfigSetPosition	3-269
ConfigureSpooler	3-271
ControlInterrupt	3-275
ConvertToSys	3-279
CopyFile	3-281
CParams	3-283
Crash	3-285
CreateAlias	3-287
CreateBigPartition	3-289
CreateContext	3-293
CreateDir	3-295
CreateExecScreen	3-297
CreateFile	3-299
CreatePartition	3-303
CreateProcess	3-307
CreateUser	3-311
CSubParams	3-313
CurrentOsVersion	3-315
DCXVersion	3-319
DeallocAliasForServer	3-321
DeallocateRods	3-323
DeallocateSegment	3-324A
DeallocExch	3-325
DeallocHugeMemory	3-326A
DeallocMemoryLL	3-327
DeallocMemorySL	3-329
DeallocRunFile	3-331
DeallocSg	3-333
DeallocUserNumbers	3-335
DebugOp	3-336A
DefineInterLevelStack	3-337
DefineLocalPageMap	3-339
DeinstallQueueManager	3-341
DeinstPSServer	3-343

Delay	3-345
DeleteByteStream	3-347
DeleteDaRecord	3-349
DeleteDir	3-351
DeleteFile	3-353
DeviceInService	3-355
DisableActionFinish	3-357
DisableCluster	3-359
DiscardInputBsC	3-361
DiscardOutputBsC	3-363
DismountVolume	3-365
DmaMapBuffer	3-367
DmaMapBufferFast	3-371
DmaUnmapBuffer	3-375
DoDirectRead	3-377
DoDirectWrite	3-381
Doze	3-385
DrainTerminalOutput	3-387
EnableSwapperOptions	3-389
EnlsAppendChar	3-391
EnlsCase	3-393
EnlsCbToCCols	3-395
EnlsClass	3-397
EnlsDeleteChar	3-401
EnlsDrawBox	3-403
EnlsDrawFormChars	3-407
EnlsDrawLine	3-409
EnlsFieldEdit	3-413
EnlsFieldEditByChar	3-419
EnlsFindC	3-423
EnlsFindRC	3-425
EnlsGetChar	3-427
EnlsGetCharWidth	3-431
EnlsGetPrevChar	3-433
EnlsInsertChar	3-435
EnlsMapCharToStdValue	3-437
EnlsMapStdValueToChar	3-439
EnlsQueryBoxSize	3-441

ErrorExit	3-443
ErrorExitString	3-447
EstablishQueueServer	3-449
Exit	3-451
ExitAndRemove	3-453
ExitListQuery	3-454A
ExitListSet	3-454C
ExpandAreaLL	3-455
ExpandAreaSL	3-457
ExpandDateTime	3-459
ExpandLocalMsg	3-461
ExpandSpec	3-465
FatalError	3-469
FComparePointer	3-471
FieldEdit	3-473
FillBufferAsyncC	3-477
FillBufferC	3-481
FillFrame	3-483
FillFrameRectangle	3-485
FillInRectangle	3-486A
FilterDebugInterrupts	3-487
FlushBufferAsyncC	3-489
FlushBufferC	3-493
FMergedOs	3-495
Format	3-497
FormatDateTime	3-499
FormatTime	3-501
FormatTimeDt	3-503
FormatTimeTm	3-505
FormEdit	3-507
ForwardRequest	3-521
FProcessorSupportsProtectedMode	3-523
FProtectedMode	3-525
FrameBackSpace	3-527
FRmos	3-529
FsCanon	3-531
FSrpUp	3-533
FValidPbCb	3-535

FVSeries	3-536A
GenResString	3-537
GetAltMsg	3-539
GetAltMsgUnexpanded	3-541
GetAltMsgUnexpandedLength	3-543
GetBoardInfo	3-545
GetBsLfa	3-547
GetBsMember	3-548A
GetCanonicalNodeAndVol	3-549
GetClstrGenerationNumber	3-550A
GetClusterId	3-551
GetClusterStatus	3-555
GetCommLineDmaStatus	3-559
GetCoprocesorStatus	3-561
GetCParasOvlyZone	3-563
GetCtosDiskPartition	3-565
GetDateTime	3-567
GetDirInfo	3-569
GetDirStatus	3-571
GetErc	3-573
GetErcLength	3-575
GetFhLongevity	3-577
GetFileErc	3-579
GetFileInfoByName	3-580A
GetFileStatus	3-581
GetFRmosUser	3-587
GetHandleStatus	3-589
GetIBusDevInfo	3-591
GetKbdId	3-595
GetKeyboardId	3-597
GetLocalDaiNumber	3-599
GetMemoryInfo	3-601
GetModuleAddress	3-603
GetModuleId	3-605
GetModuleResourceInfo	3-606A
GetMsg	3-607
GetMsgUnexpanded	3-609
GetMsgUnexpandedLength	3-611

GetNlsDateName	3-613
GetNlsDateTimeTemplate	3-617
GetNlsKeycapText	3-619
GetNlsTable	3-621
GetOvlyStats	3-623
GetPartitionExchange	3-625
GetPartitionHandle	3-627
GetPartitionStatus	3-629
GetPartitionSwapMode	3-633
GetPAscb	3-635
GetPNlsTable	3-637
GetProcInfo	3-639
GetPSCounters	3-643
GetPStructure	3-645
GetQMStatus	3-655
GetRouteTable	3-659
GetRsLfa	3-663
GetScsiInfo	3-665
GetSegmentLength	3-667
GetServerMsg	3-669
GetSlotInfo	3-671
GetStamFileHeader	3-675
GetStandardErcMsg	3-677
GetSysCmdInfo	3-679
GetUcb	3-681
GetUserFileEntry	3-683
GetUserLocation	3-687
GetUserNumber	3-689
GetUserStatus	3-691
GetVhb	3-695
GetWsUserName	3-697
HeapAlloc	3-699
HeapFree	3-701
HeapInit	3-703
InitAltMsgFile	3-705
InitCharMap	3-707
InitCommLine	3-709
InitErcFile	3-719

InitLargeOverlays	3-721
InitLocalPageMap	3-723
InitMsgFile	3-725
InitOverlays	3-727
InitSysCmds	3-729
InitVidFrame	3-731
InstallNet	3-735
KeyboardProfile	3-737
KillProcess	3-739
LaFromP	3-740A
LaFrom	3-740C
LibFree	3-740E
LibGetHandle	3-740G
LibGetInfo	3-740I
LibGetProcInfo	3-740K
LibLoad	3-740M
LoadBackgroundPalette	3-741
LoadColorStyleRam	3-743
LoadFontRam	3-745
LoadInteractiveTask	3-749
LoadPrimaryTask	3-751
LoadRunFile	3-755
LoadTask	3-757
LockIn	3-761
LockInContext	3-763
LockOut	3-765
LockVideo	3-767
LockVideoForModify	3-769
LockXbis	3-771
LogMsgIn	3-773
LogRespond	3-775
LogRequest	3-777
LookUpField	3-779
LookUpNumber	3-781
LookUpReset	3-783
LookUpString	3-785
LookUpValue	3-786A
MakePermanent	3-787

MakePermanentP	3-789
MakeRecentlyUsed	3-791
MapBusAddress	3-793
MapCsIOvly	3-797
MapIOvlyCs	3-799
MapPhysicalAddress	3-801
MapPStubPProc	3-805
MapXBusWindow	3-807
MapXBusWindowLarge	3-811
MarkKeyedQueueEntry	3-815
MarkNextQueueEntry	3-819
McrVersion	3-821
MediateIntHandler	3-823
MenuEdit	3-825
Mode3DmaReload	3-833
MountVolume	3-835
MouseVersion	3-837
MoveFrameRectangle	3-839
MoveOverlays	3-841
NameAllocClass	3-842A
NameQuery	3-842C
NameRegister	3-842E
NameRemove	3-842G
NewProcess	3-843
NlsCase	3-847
NlsClass	3-849
NlsCollate	3-851
NlsFormatDateTime	3-855
NlsGetYesNoStrings	3-859
NlsGetYesNoStringSize	3-861
NlsNumberAndCurrency	3-863
NlsParseTime	3-865
NlsSpecialCharacters	3-867
NlsStdFormatDateTime	3-869
NlsULCmpB	3-873
NlsVerifySignatures	3-875
NlsYesNoOrBlank	3-877
NlsYesOrNo	3-879

NPrint	3-881
ObtainAccessInfo	3-883
ObtainUserAccessInfo	3-887
OldReadHardId	3-891
OldWriteHardId	3-893
OpenByteStream	3-895
OpenByteStreamC	3-897
OpenDaFile.....	3-899
OpenFile	3-901
OpenFileLL	3-905
OpenNlsFile	3-909
OpenPSLogSession	3-911
OpenPSStatSession	3-913
OpenRsFile	3-915
OpenRtClock	3-917
OpenServerMsgFile	3-919
OpenTerminal.....	3-921
OpenUserFile	3-925
OpenVidFilter.....	3-927
OsVersion	3-929
OutputBytesWithWrap	3-933
OutputQuad	3-935
OutputToVid0.....	3-937
OutputWord	3-939
PaFromP	3-941
PaFromSn	3-943
ParseFileSpec	3-945
ParseSpecForDir	3-951
ParseSpecForFile	3-955
ParseSpecForNode.....	3-957
ParseSpecForPassword	3-961
ParseSpecForVol	3-965
ParseTime	3-969
PDAssignMouse	3-971
PDGetCursorPos	3-973
PDGetCursorPosNSC	3-975
PDInitialize	3-977
PDLoadCursor	3-979

PDLoadSystemCursor	3-985
PDQueryControls	3-987
PDQuerySystemControls	3-989
PDReadCurrentCursor	3-991
PDReadIconFile	3-995
PDSetCharMapVirtualCoordinates	3-997
PDSetControls	3-999
PDSetCursorDisplay	3-1001
PDSetCursorPos	3-1003
PDSetCursorPosNSC	3-1005
PDSetCursorType	3-1007
PDSetMotionRectangle	3-1009
PDSetMotionRectangleNSC	3-1011
PDSetSystemControls	3-1013
PDSetTracking	3-1015
PDSetVirtualCoordinates	3-1017
PDTranslateNSCToVC	3-1019
PDTranslateVCToNSC	3-1021
PosFrameCursor	3-1023
PostKbdTable	3-1025
PrintAltMsg	3-1029
PrintErc	3-1031
PrintFileClose	3-1033
PrintFileOpen	3-1035
PrintFileStatus	3-1037
PrintMsg	3-1039
ProgramColorMapper	3-1041
PSCloseSession	3-1045
PSDeinstServer	3-1047
PSend	3-1049
PSGetCounters	3-1051
PSOpenLogSession	3-1055
PSOpenStatSession	3-1059
PSReadLog	3-1063
PSResetCounters	3-1067
PurgeMcr	3-1069
PutBackByte	3-1071
PutByte	3-1073

PutChar	3-1075
PutCharsAndAttrs	3-1077
PutFrameAttrs	3-1079
PutFrameChars	3-1081
PutFrameCharsAndAttrs	3-1083
PutPointer	3-1085
PutQuad	3-1087
PutWord	3-1089
QueryBigMemAvail	3-1091
QueryBoardInfo	3-1093
QueryBounds	3-1097
QueryCharsAndAttrs	3-1099
QueryCoproprocessor	3-1101
QueryCursor	3-1103
QueryDaLastRecord	3-1105
QueryDaRecordStatus	3-1107
QueryDcb	3-1109
QueryDefaultRespExch	3-1111
QueryDeviceName	3-1113
QueryDeviceNames	3-1117
QueryDiskGeometry	3-1119
QueryExitRunFile	3-1125
QueryFrameAttrs	3-1127
QueryFrameBounds	3-1129
QueryFrameChar	3-1131
QueryFrameCharsAndAttrs	3-1133
QueryFrameCursor	3-1135
QueryFrameString	3-1137
QueryIOOwner	3-1139
QueryKbdLeds	3-1143
QueryKbdState	3-1145
QueryLdtR	3-1147
QueryMail	3-1149
QueryMemAvail	3-1151
QueryModulePosition	3-1153
QueryNodeName	3-1155
QueryPagingStatistics	3-1156A
QueryProcessInfo	3-1157

QueryProcessNumber	3-1161
QueryRequestInfo	3-1163
QueryTrapHandler	3-1165
QueryUserLocation	3-1169
QueryVidBs	3-1171
QueryVideo	3-1173
QueryVidHdw	3-1175
QueryWsNum	3-1183
QueryZoomBoxPosition	3-1185
QueryZoomBoxSize	3-1187
QueueMgrVersion	3-1189
Read	3-1191
ReadActionCode	3-1193
ReadActionKbd	3-1195
ReadAsync	3-1197
ReadBsRecord	3-1199
ReadByte	3-1201
ReadBytes	3-1203
ReadByteStreamParameterC	3-1205
ReadCommLineStatus	3-1207
ReadDaFragment	3-1209
ReadDaRecord	3-1211
ReadDirSector	3-1213
ReadHardId	3-1215
ReadInputEvent	3-1217
ReadInputEventNSC	3-1223
ReadKbd	3-1225
ReadKbdDataDirect	3-1227
ReadKbdDirect	3-1233
ReadKbdInfo	3-1237
ReadKbdStatus	3-1243
ReadKeyedQueueEntry	3-1247
ReadKeySwitch	3-1251
ReadMcr	3-1253
ReadNextQueueEntry	3-1257
ReadOsKbdTable	3-1259
ReadPSLog	3-1263
ReadRemote	3-1265

ReadRsRecord	3-1267
ReadStatusC	3-1269
ReadTerminal	3-1271
ReadToNextField	3-1275
ReallocHugeMemory	3-1276A
ReceiveCommLineDma	3-1277
ReinitLargeOverlays	3-1279
ReinitOverlays	3-1281
ReinitStubs	3-1283
ReleaseByteStream	3-1285
ReleaseByteStreamC	3-1287
ReleasePermanence	3-1289
ReleaseRsFile	3-1291
RemakeAliasForServer	3-1293
RemakeFh	3-1295
RemapBusAddress	3-1299
RemoteBoot	3-1303
RemoveFfsBrackets	3-1307
RemoveKeyedQueueEntry	3-1309
RemoveMarkedQueueEntry	3-1311
RemovePartition	3-1313
RemoveQueue	3-1315
RenameByteStream	3-1317
RenameFile	3-1319
ReopenFile	3-1321
Request	3-1323
RequestDirect	3-1325
RequestRemote	3-1327
RescheduleMarkedQueueEntry	3-1329
RescheduleProcess	3-1331
ReserveBusAddress	3-1333
ReservePartitionMemory	3-1335
ResetCommLine	3-1339
ResetDeviceHandler	3-1341
ResetFrame	3-1343
ResetIBusHandler	3-1345
ResetMemoryLL	3-1347
ResetPSCounters	3-1349

ResetTimerInt	3-1351
ResetTrapHandler	3-1353
ResetVideo	3-1355
ResetVideoGraphics	3-1359
ResetXBusMIsr	3-1365
ResizeIoMap	3-1367
ResizeSegment	3-1368A
Respond	3-1369
ResumeContext	3-1371
ReuseAlias	3-1373
ReuseAliasLarge	3-1375
RewriteMarkedQueueEntry	3-1377
RgParam	3-1379
RgParamInit	3-1381
RgParamSetEltNext	3-1383
RgParamSetListStart	3-1385
RgParamSetSimple	3-1387
RkvsVersion	3-1389
RsrcCopyFromFile	3-1390A
RsrcCopyFromRsrcSet	3-1390E
RsrcCopyRestRunFile	3-1390G
RsrcDelete	3-1390I
RsrcEndSetAccess	3-1390K
RsrcGetAllSetTypeInfo	3-1390M
RsrcGetCountAllSetTypes	3-1390O
RsrcGetCountSetType	3-1390Q
RsrcGetDesc	3-1390S
RsrcGetSetTypeInfo	3-1390U
RsrcInitSetAccess	3-1390W
RsrcSessionEnd	3-1390Z1
RsrcSessionInit	3-1390Z3
SbPrint	3-1391
ScanToGoodRsRecord	3-1393
ScreenPrintVersion	3-1395
ScrollFrame	3-1397
ScrubFile	3-1399
ScsiCdbDataIn	3-1401
ScsiCdbDataInAsync	3-1405

ScsiCdbDataOut	3-1409
ScsiCdbDataOutAsync	3-1413
ScsiClosePath	3-1417
ScsiManagerNameQuery	3-1419
ScsiOpenPath	3-1421
ScsiQueryInfo	3-1427
ScsiQueryPathParameters	3-1433
ScsiRequestSense	3-1437
ScsiReset	3-1441
ScsiSetPathParameters	3-1443
ScsiTargetCdbCheck	3-1447
ScsiTargetCdbWait	3-1449
ScsiTargetDataReceive	3-1451
ScsiTargetDataReceiveAsync	3-1455
ScsiTargetDataTransmit	3-1457
ScsiTargetDataTransmitAsync	3-1461
ScsiTargetOperationsAbort	3-1463
ScsiWaitCdbAsync	3-1465
ScsiWaitTargetDataAsync	3-1467
SemClear	3-1468A
SemClearCritical	3-1468C
SemClearProcessLocks	3-1468E
SemClose	3-1468G
SemEnumerate	3-1468I
SemEnumerateWaiting	3-1468K
SemLock	3-1468O
SemLockCritical	3-1468Q
SemMuxWait	3-1468S
SeNotify	3-1468W
SemOpen	3-1468Y
SemQuery	3-1468Z3
SemQueryProcessLock	3-1468Z9
SemSet	3-1468Z11
SemWait	3-1468Z13
SendBreakC	3-1471
SeqAccessCheckpoint	3-1473
SeqAccessClose	3-1475
SeqAccessControl	3-1477

SeqAccessDiscardBufferData	3-1485
SeqAccessModeQuery	3-1487
SeqAccessModeSet	3-1491
SeqAccessOpen	3-1495
SeqAccessRead	3-1505
SeqAccessRecoverBufferData	3-1509
SeqAccessStatus	3-1513
SeqAccessVersion	3-1517
SeqAccessWrite	3-1521
SerialNumberQuery	3-1525
SerialNumberOldOsQuery	3-1527
ServerRq	3-1529
Set386TrapHandler	3-1531
SetAlphaColorDefault	3-1533
SetBsLfa	3-1535
SetDaBufferMode	3-1537
SetDateTime	3-1539
SetDateTimeMode	3-1541
SetDefault386TrapHandler	3-1542A
SetDefaultTrapHandler	3-1543
SetDeltaPriority	3-1545
SetDeviceHandler	3-1547
SetDevParams	3-1549
SetDirStatus	3-1553
SetDiskGeometry	3-1557
SetDispMsw287	3-1561
SetExitRunFile	3-1563
SetFhLongevity	3-1565
SetField	3-1567
SetFieldNumber	3-1569
SetFileStatus	3-1571
SetIBusHandler	3-1577
SetImageMode	3-1581
SetImageModeC	3-1583
SetIntHandler	3-1585
SetIOOwner	3-1589
SetKbdLed	3-1593
SetKbdUnencodedMode	3-1597

SetKeyboardOptions	3-1599
SetLdtRDs	3-1603
SetLPIsr	3-1605
SetModuleId	3-1606A
SetMsgRet	3-1607
SetNode	3-1609
SetPartitionExchange	3-1611
SetPartitionLock	3-1613
SetPartitionName	3-1615
SetPartitionSwapMode	3-1617
SetPath	3-1621
SetPrefix	3-1623
SetPStructure	3-1625
SetRsLfa	3-1629
SetScreenVidAttr	3-1631
SetSegmentAccess	3-1633
SetStyleRam	3-1637
SetStyleRamEntry	3-1639
SetSwapDisable	3-1641
SetSysInMode	3-1643
SetTerminal	3-1647
SetTimerInt	3-1653
SetTrapHandler	3-1655
SetUpMailNotification	3-1657
SetUserFileEntry	3-1659
SetVideoTimeout	3-1663
SetWsUserName	3-1665
SetXBusMIsr	3-1667
SgFromSa	3-1671
ShortDelay	3-1675
ShrinkAreaLL	3-1677
ShrinkAreaSL	3-1679
ShrinkPartition	3-1681
SignOnLog	3-1683
SnFromSr	3-1687
SpoolerPassword	3-1689
SpoolerVersion	3-1691
SrFromSn	3-1693

StatisticsVersion	3-1695
StringsEqual	3-1697
SuspendProcess	3-1699
SuspendUser	3-1701
SwapContextUser	3-1703
SwapInContext	3-1705
SwappingRequests	3-1707
SwapXBusEar	3-1709
SystemCommonCheck	3-1711
SystemCommonConnect	3-1713
SystemCommonInstall	3-1715
SystemCommonQuery	3-1719
TerminateAllOtherContexts	3-1721
TerminateContext	3-1723
TerminateContextUser	3-1725
TerminatePartitionTasks	3-1727
TerminateQueueServer	3-1729
TerminationRequests	3-1731
TextEdit	3-1733
TransmitCommLineDma	3-1737
TruncateDaFile	3-1739
TsConnect	3-1741
TsDataChangeParams	3-1745
TsDataCheckpoint	3-1747
TsDataCloseLine	3-1749
TsDataOpenLine	3-1751
TsDataRead	3-1755
TsDataRetrieveParams	3-1757
TsDataUnacceptCall	3-1759
TsDataWrite	3-1761
TsDeinstall	3-1763
TsDial	3-1765
TsDoFunction	3-1767
TsGetStatus	3-1771
TsHold	3-1773
TsLoadCallProgressTones	3-1775
TsOffHook	3-1777
TsOnHook	3-1779

TsQueryConfigParams	3-1781
TsReadTouchTone	3-1783
TsRing	3-1787
TsSetConfigParams	3-1789
TsVersion	3-1791
TsVoiceConnect	3-1793
TsVoicePlaybackFromFile	3-1795
TsVoiceRecordToFile	3-1799
TsVoiceStop	3-1803
TurnOffMailNotification	3-1805
ULCmpB	3-1807
UndeleteFile	3-1809
UnlockInContext	3-1811
UnlockVideo	3-1813
UnlockVideoForModify	3-1815
UnlockXbis	3-1817
UnmapBusAddress	3-1819
UnmapPhysicalAddress	3-1821
UnmarkQueueEntry	3-1823
UnsuspendProcess	3-1825
UnsuspendUser	3-1827
UnzoomBox	3-1829
UpdateOverlayLru	3-1831
VacatePartition	3-1833
Wait	3-1835
WaitLong	3-1837
WhereTerminalBuffer	3-1839
WildcardClose	3-1843
WildcardInit	3-1845
WildcardMatch	3-1847
WildcardNext	3-1849
Write	3-1851
WriteAsync	3-1853
WriteBsRecord	3-1855
WriteByte	3-1857
WriteByteStreamParameterC	3-1859
WriteCommLineStatus	3-1861
WriteDaFragment	3-1863

WriteDaRecord	3-1865
WriteHardId	3-1867
WriteIBusDevice	3-1869
WriteIBusEvent	3-1871
WriteKbdBuffer	3-1875
WriteLog	3-1879
WriteRemote	3-1881
WriteRsRecord	3-1883
WriteStatusC	3-1885
XbifVersion	3-1887
ZoomBox	3-1889
ZPrint	3-1893

4 System Structures

Chapter Organization	4-1
System Structures Table of Contents	4-2
Allowed States Table	4-5
Application System Control Block	4-9
Batch Control Block	4-17
Cache Entry Descriptor	4-23
Cache Pool Descriptor	4-27
Chords Table	4-31
Command Masks Table	4-37
Communications Configuration Descriptor	4-41
Communications Status Buffer	4-47
Conditions Table	4-67
Control Chords Table	4-71
CPU Description Table	4-75
CPU Description Table (pre-3.0)	4-87
Data Control Structure	4-95
Decoding Offsets Table	4-101
Decoding Table	4-105
Device Control Block	4-109
Device Control Block (pre-3.0)	4-117
Diacritic Masks Table	4-127
Diacritics Table	4-131
Directory Entry in Master File Directory Block	4-135
Emulation Data Block Header	4-139

Emulation LEDs Table	4-145
Emulation Table	4-149
Expanded Date/Time Format	4-153
Extended Partition Descriptor	4-157
Extended Process Control Block	4-165
File Entry in a Directory Block	4-169
File Header Block	4-173
Frame Descriptor	4-183
High Sierra Directory Record	4-189
ISO Directory Record	4-191
Log File Record Format	4-193
Low Memory Allocation	4-213
Master File Directory Block	4-219
Multibyte String Masks Table	4-222A
Multibyte String Offsets Table	4-222E
Multibyte Strings Table	4-222I
Operating System Flags	4-223
Partition Configuration Block	4-229
Partition Descriptor	4-233
Partition Information Block	4-239
Partition Keyboard Information Structure	4-243
Performance Statistics Structure	4-251
Port Structure	4-257
Process Control Block	4-271
Queue Entry Header	4-275
Queue File Header	4-279
Queue Status Block	4-285
Repeat Attributes Table	4-289
Resource Descriptor	4-292A
Special Keys Table	4-293
Spooler Control Queue Entry	4-299
Spooler Scheduling Queue Entry	4-303
Spooler Status Queue Entry	4-309
Standard File Header Format	4-315
Standard Record Header Format	4-319
Standard Record Trailer Format	4-323
String Masks Table	4-327
String Offsets Table	4-331

Strings Table	4-335
System Configuration Block	4-339
System Date/Time Structure	4-347
Table Descriptor Array	4-351
Telephone Service Configuration Block	4-355
Telephone Service Configuration File Format	4-361
Telephone Status Structure	4-365
Terminal Output Buffer	4-375
Timer Pseudointerrupt Block	4-379
Timer Request Block Format	4-383
Translation Data Block Header	4-387
Translation Table	4-393
User Control Block	4-397
Variable Length Parameter Block	4-401
Video Control Block	4-405
Voice Control Structure	4-413
Voice File Header	4-419
Voice File Record	4-425
Voice Processor Control Block	4-429
Volume Home Block	4-437
Volume Home Block (pre-3.0)	4-451
Appendix A	
Status Codes	A-1
Appendix B	
Standard Character Set	B-1
Appendix C	
Keyboard Codes	C-1
Appendix D	
Request Codes in Numeric Sequence	D-1
Appendix E	
NLS Templates	E-1
Appendix F	
Message File Macro Definitions	F-1

Appendix G	
X-Bus and I-Bus Module IDs	G-1
Appendix H	
Byte Stream Escape Sequences	H-1
Appendix I	
System Common Procedure Parameter Syntax	I-1
Appendix J	
Operation Data	J-1
Appendix K	
Resource Types	K-1

List of Figures

2-1.	Operations by Function	2-2
3-1.	Form Showing a String Response	3-508
3-2.	Form Showing Yes or No Choice Responses	3-508
3-3.	Menu With One Item Selected	3-825
3-4.	Menu With Three Items Selected	3-826
3-5.	User-Defined Graphics Cursor	3-981
3-6.	Six Rows of Sample Graphics Cursor Loaded into Screen Memory	3-981
3-7.	Pascal Code Defining the Left-Arrow Cursor	3-982

List of Tables

ATM-1	Common Prefixes	xli
ATM-2	Common Roots	xliB
B-1.	Standard Character Set	B-2
B-2.	Graphic Representation of the Standard Character Set	B-13
B-3.	Nationalized Video Display Characters	B-15
C-1.	Keyboard Codes Generated by an Unencoded K1 Keyboard	C-2
C-2.	Keyboard Codes: Key Differences for K2, K3, K5, and SG101-K keyboards	C-6
C-3.	Keyboard Codes Generated by an Unencoded SG102-K Keyboard	C-7
E-1.	NLS Templates	E-2
G-1.	Module IDs	G-5
H-2.	Escape Sequence Redirecting a Video Byte Stream ..	H-1
H-3.	Escape Sequence Controlling Character Attributes ..	H-2
H-4.	Escape Sequence Controlling Pause	H-4
H-5.	Escape Sequence Controlling Scrolling and Cursor Positioning	H-5
H-6.	Escape Sequence Performing Miscellaneous Functions	H-6
H-7.	Video Byte Stream Interpretation of Special Characters	H-8

H-8.	Submit File Escape Sequences	H-9
J-1.	Kernel Primitives	J-2
J-2.	Requests	J-3
J-3.	System Common Procedures	J-17
J-4.	Standard Operating System Library Procedures	J-21
J-5.	Mouse Library Procedures	J-30
J-6.	Asynchronous System Service Library Procedures...	J-31
K-1.	Resource Type Codes	K-1

Chapter Organization

This chapter arranges the system structures alphabetically by name. The name appears at the top of each page on which the structure is described.

If you don't know the name of the structure you are looking for but know generally what it does, you can look up the name in the table of contents starting on the next page of this chapter. The table of contents groups the structures by their general function, such as file system or video.

The introduction to each structure provides a general description of the structure. Typically the introduction points out the relevance of the structure to a system or application programmer, explains how the structure can be accessed programmatically, provides references to documentation where more information can be obtained, and lists all the other related structures in this chapter.

Following the introduction, the structure fields are described in tabular form. In most cases, this presentation gives as much information about the structure as you would need at a glance. If you would like additional information, you can consult the detailed field descriptions following the table.

Tables describing video and submit file byte stream escape sequences are contained in Appendix H.

System Structures Table of Contents

Cacher Structures

Cache Pool Descriptor	4-27
Cache Entry Descriptor	4-23

CD-ROM Structures

High Sierra Directory Record	4-189
ISO Directory Record	4-191

Cluster Structure

Communications Status Buffer	4-47
------------------------------------	------

Communications Structure

Communications Configuration Descriptor	4-41
---	------

File Structures

Device Control Block	4-109
Device Control Block (pre-3.0)	4-117
Directory Entry in a Master File Directory Block	4-135
File Entry in a Directory Block	4-169
Master File Directory Block	4-219
File Header Block	4-173
Log File Record Format	4-193
Standard File Header Format	4-315
Standard Record Header Format	4-319
Standard Record Trailer Format	4-323
User Control Block	4-397
Volume Home Block	4-437
Volume Home Block (pre-3.0)	4-451

Keyboard Structures

Allowed States Table	4-5
Chords Table	4-31
Command Masks Table	4-37

Keyboard Structures (Continued)

Conditions Table	4-67
Control Chords Table	4-71
Decoding Offsets Table	4-101
Decoding Table	4-105
Diacritic Masks Table	4-127
Diacritics Table	4-131
Emulation Data Block Header	4-139
Emulation LEDs Table	4-145
Emulation Table	4-149
Multibyte String Masks Table	4-222A
Multibyte String Offsets Table	4-222E
Multibyte Strings Table	4-222I
Partition Keyboard Information Structure	4-243
Repeat Attributes Table	4-289
Special Keys Table	4-293
String Masks Table	4-327
String Offsets Table	4-331
Strings Table	4-335
Table Descriptor Array	4-351
Translation Data Block Header	4-387
Translation Table	4-393

Partition Structures

Application System Control Block	4-9
Batch Control Block	4-17
Extended Partition Descriptor	4-157
Partition Configuration Block	4-229
Partition Descriptor	4-233
Partition Information Block	4-239
Variable Length Parameter Block	4-401

Performance Statistics Structure

Performance Statistics Structure	4-251
--	-------

Queue Manager Structures	
Queue Entry Header	4-275
Queue File Header	4-279
Queue Status Block	4-285
 Resource Management Structures	
Resource Descriptor	4-292A
 Shared Resource Processor Structures	
CPU Description Table	4-75
CPU Description Table (pre-3.0)	4-87
Terminal Output Buffer	4-375
 Spooler Queue Entries	
Spooler Control Queue Entry	4-299
Spooler Scheduling Queue Entry	4-303
Spooler Status Queue Entry	4-309
 System Structures	
Extended Process Control Block	4-165
Port Structure	4-257
Process Control Block	4-271
System Configuration Block	4-339
 Timer and Interrupt Handler Structures	
Expanded Date/Time Format	4-153
Low Memory Allocation	4-213
System Date/Time Structure	4-347
Timer Pseudointerrupt Block	4-379
Timer Request Block Format	4-383
 Video Structures	
Frame Descriptor	4-183
Video Control Block	4-405

Voice/Data Structures	
Data Control Structure	4-95
Telephone Service Configuration	
Block (TSCB).....	4-355
Telephone Service Configuration	
File Format	4-361
Telephone Status Structure.....	4-365
Voice Control Structure	4-413
Voice File Header	4-419
Voice File Record	4-425
Voice Processor Control Block (VPCB).....	4-429

This page intentionally left blank.

Description

The *Allowed States Table* is used by the keyboard process to identify chords that can affect the emulation or translation of a given key.

Each entry in the table is a word that contains data for a single key. The first entry corresponds to the key with a keyboard code of 0; the second entry corresponds to the key with a keyboard code of 1, and so forth.

Each entry contains the identifiers of the chords that affect emulation or translation of the key. For example, if entry 40 contains a mask value of 3, the key with a keyboard code of 40 is affected by the chords with identifiers 1 and 2. For an explanation of "chord identifier," see the Chords Table.

The Allowed States Table is a support table that resides in a translation or emulation data block. The *oAllowedStates* field in the Translation or Emulation Data Block Header contains the offset, from the beginning of the data block, of the Allowed States Table. The *cKeys* field of either header contains the size, in words, of the Allowed States Table.

To create or modify an allowed states table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Allowed States Table

Offset	Field*	Size (bytes)	Description
0	key0	2	Chord identifiers for key 0
2	key1	2	Chord identifiers for key 1
.			
.			
.			

*The Allowed States Table contains a variable number of fields.

key0

is a value that contains the identifiers for chords that influence the key with a keyboard or character code of 0.

key1

is a value that contains the identifiers for chords that influence the key with a keyboard or character code of 1.

.

.

.

and so forth.

This page intentionally left blank.

Description

The *Application System Control Block (ASCB)* is one of several structures comprising the User Structure of an application partition. (For a description of the User Structure, see "Partitions and Partition Management" in the *CTOS Operating System Concepts Manual*.) Fields in the ASCB describe the status of the Executive, Batch, and other applications that currently may be running in the partition.

A pointer to this structure is returned by GetPAscbl or by GetPStructure (code 250h).

Related Partition Structures

- Batch Control Block
- Extended Partition Descriptor
- Partition Configuration Block
- Partition Descriptor
- Variable Length Parameter Block

This page intentionally left blank

(continued)

Application System Control Block

Offset	Field	Size (bytes)	Description
0	fhSwapFile	2	identifies a run file with overlays
2	pVLPB	4	VLPB address in the long-lived memory
6	fExecScreen	1	preserves the Executive's screen setup
7	fChkBoot	1	TRUE means the Executive is loaded
8	ercRet	2	ErcTermination parameters
10	pbMsgRet	4	address of a string in long-lived memory
14	cbMsgRet	2	size of the string
16	reserved	6	
22	fTermination	1	TRUE if Action-Finish tried when disabled
23	fVacate	1	is not used
24	saLastTask	2	is the segment address of the run file base
26	fExecFont	1	TRUE means the default font is loaded
27	bActionCode	1	contains the last action code detected.
28	cParMemArray	2	is the primary task memory array size
30	rgbReservedforBatch1	15	is reserved for use by Batch
45	rgbReservedforMail	19	is reserved for use by Mail
64	sbUserName	31	is the current workstation user name
95	sbPassword	13	is the SignOn user password
108	sbCmdFile	79	is the user's Executive command file.name
187	rgbReservedforBatch2	93	is reserved for use by Batch
280	fColor	1	TRUE means color is supported
281	rgbColorBytes	8	area for saving current color palette
289	fReverseVideo	1	TRUE if the screen is in reverse video
290	fBackground	1	TRUE if background color is set
291	bBackground	1	specifies background color set
292	fVidFilter	1	TRUE if the video filter is active
293	bStatusRet	1	a secondary status indicator for C runtime
294	reserved2	10	reserved for future use

fhSwapFile

is a file handle of the run file containing the code segment overlay currently running in this application partition. If the current run file is not using overlays, *fhSwapFile* is 0FFFFh. The Chain operation sets the value of this field. (For details, see the description of Chain in Chapter 3 in this manual.)

pVLPB

is the memory address of the VLPB in the long-lived memory of the application partition.

fExecScreen

is a flag that is set to TRUE to preserve the Executive's screen setup (frames 0, 1, 2, and so forth). The ResetVideo operation sets this flag to FALSE. If *fExecScreen* is FALSE when the Executive is loaded, the Executive reinitializes the video subsystem.

fChkBoot

is a flag. TRUE means the Executive has been loaded since the system was bootstrapped. The flag is FALSE during operating system initialization until the time the Executive is loaded.

ercRet

is a word into which the Chain and ErrorExit operations write their *ercTermination* parameters.

pbMsgRet

cbMsgRet

describe a message (text string) which is stored in the long-lived memory of an application partition. The message is created by an application program running in the partition so that the message can be displayed by the Executive when the Executive is loaded into the partition.

fTermination

is set to TRUE when a user tries to **Action-Finish** an application program when **Action-Finish** is disabled; or when an application program tries to terminate the program in a locked secondary partition. This is set to FALSE when a program replaces the old program in the partition.

fVacate

is a flag that is set to TRUE when a user or an application program tries to vacate the program in a locked secondary partition. (Note that *fVacate* is not set to TRUE if the partition to be vacated is swapped to disk). *fVacate* is set to FALSE when a program replaces the old program in the partition.

saLastTask

is the segment address of the base of the run file loaded into the partition.

fExecFont

is a flag. TRUE means the default system font is loaded. The operating system sets *fExecFont* to FALSE when the font is changed. When the Executive finds *fExecFont* set to FALSE, it reloads the default font and resets the flag.

bActionCode

Contains the last action code detected by the keyboard process (not including the codes for **Action-A**, **Action-B**, or **Action-Finish**). The `ReadActionCode` and `ReadActionKbd` operations can be called to obtain this same information.

cParMemArray

is the size of the memory array (in 16-byte paragraphs) of the primary task (first run file loaded into the partition) when the task is loaded. Secondary tasks (subsequent run files loaded into the same partition) cannot use a memory array.

rgbReservedforBatch1

is an array of bytes reserved for use by Batch.

rgbReservedforMail

is an array of bytes reserved for use by electronic Mail.

sbUserName

is the current workstation user name (the first byte of the string is the length of the name).

sbPassword

is the password the user gave when signing on (for accessing the user configuration file).

sbCmdFile

is the name of the user's Executive command file.

rgbReservedforBatch2

is an array of bytes reserved for use by Batch.

fColor

is a flag that is TRUE if the workstation supports color.

rgbColorBytes

is an area in which the Executive can save the current color palette.

fReverseVideo

is a flag that is TRUE if the partition's screen is in reverse video.

fBackground

is a flag that is TRUE if the partition's screen is set to use background color.

bBackground

is the value of the background color set.

fVidFilter

is a flag that is TRUE if a filter is redirecting the video byte stream output of a program to which the caller chains to the temporary file [Scr]<\$>bsFilter.tmp.

bStatusRet

stores the status byte passed to the exit routine so that the status value can be made available to the next application.

This page intentionally left blank

Description

The *Batch Control Block* stores Batch Manager control information while the Batch Manager chains to a different run file in the application partition. The information is made available again when the Batch Manager is loaded into the partition and is ready to resume execution. Because the Batch Control Block is part of the User Structure, the contents are preserved, even when long-lived memory is reset. (For details on Batch, see the *CTOS Batch Manager II Installation, Configuration, and Programming Guide*.)

The Batch Control Block is an optional User Structure component that can be specified when the partition is created using the CreatePartition operation.

Related Partition Structures

- Application System Control Block
- Extended Partition Descriptor
- Partition Configuration Block
- Partition Descriptor
- Variable Length Parameter Block

This page intentionally left blank

(continued)

Batch Control Block

Offset	Field	Size (bytes)	Description
0	SysInBuffer	512	buffer for reading from SysIn
512	SysOutBuffer	512	buffer for writing to SysOut
1024	sbJobName	13	job name specified in a \$Job statement
1037	sbQueueName	51	queue name specified in Batch command
1088	fSysInBsOpen	1	TRUE if the SysIn file is open
1089	SysInBswa	50	SysIn byte stream work area
1139	fSysOutBsOpen	1	TRUE if the SysOut file is open
1140	SysOutBswa	51	SysOut byte stream work area
1191	fhLogLL	2	Batch manager LL log file handle
1193	reserved	4	
1197	qeh	4	Batch communication queue entry handle
1201	qehStatus	4	queue status for the current qeh
1205	bSequence	1	order that Batch Manager was installed
1206	dateTime	4	date and time the SysOut file is created
1210	bSpvsrCmnd	1	indicates action to take on a Batch job
1211	UniqueQuad	4	date/time identifying a specific Batch job
1215	timeStarted	4	time a Batch job was started
1219	sbSysOutSpoolQueue	51	spooler queue name (default = Spl)
1270	cLogFileCopies	1	number of copies of Log file to print
1271	fTruncateLogFile	1	TRUE indicates Log file can be truncated
1272	JobStepPriority	1	priority of a Batch command (default = 129)

SysInBuffer

is a buffer used for reading from SysIn.

SysOutBuffer

is a buffer used for writing to SysOut.

sbJobName

is the job name specified in the \$Job statement of a JCL file. The first byte is the size of the Job name string.

sbQueueName

is the queue name specified in the Batch command or is the default queue name Batch. The first byte is the size of the queue name string.

fSysInBsOpen

is a flag that is TRUE if the SysIn file is open.

SysInBswa

is the Byte Stream Work Area associated with the SysIn.

fSysOutBsOpen

is a flag that is TRUE if the SysOut file is open.

SysOutBswa

is the Byte Stream Work Area associated with the SysOut.

fhLogLL

is the file handle associated with the long-lived log file for a Batch manager.

qeh

is the queue entry handle used for communication between the Batch Supervisor and Batch Manager.

qehStatus

is the queue status for the current queue entry handle.

bSequence

is a byte that describes the order in which the Batch Manager was installed.

dateTime

indicates the date and time at which the SysOut file is created.

bSpvsrCmnd

is a byte that indicates to the Batch Supervisor what action to take on a given batch job. The command is sent from **Batch Status**.

UniqueQuad

is a unique date/time stamp that identifies a specific Batch job.

timeStarted

specifies the time that the batch job was started.

sbSysOutSpoolQueue

is the spooler queue name (default = Spl). The first byte is the size of the spooler queue name string.

cLogFileCopies

indicates the number of copies of the Log file to print.

fTruncateLogFile

is a flag that is TRUE if the Log file is to be truncated.

JobStepPriority

is the priority specified in the **Batch** command or is the default priority 129.

Related Partition Structures:

Application System Control Block

Extended Partition Descriptor

Partition Configuration Block

Partition Descriptor Block

Variable Length Parameter Block

This page intentionally left blank

Description

For each cache entry in a cache pool, there is a *Cache Entry Descriptor* that contains all the information unique to the entry. Each descriptor includes the memory addresses of the next and previous entries in the LRU list and the hash queue, the status and identification number of this cache entry, and the memory address of the data buffer for this entry.

(For details on the cacher, see the *CTOS Operating System Concepts Manual*.)

Related Memory Management Structures

Cache Pool Descriptor

This page intentionally left blank

(continued)

Cache Entry Descriptor

Offset	Field	Size (bytes)	Description
0	pLruListNext	4	address of next entry in the LRU list
4	pLruListPrev	4	address of previous entry in the LRU list
8	pDataBuffer	4	memory address of this entry's data buffer
12	pHashQueueNext	4	address of next entry in the hash queue
16	pHashQueuePrev	4	address of previous entry in hash queue
20	cacheID	8	unique ID associated with this cache entry
28	statusFlags	2	internally used status flags
29	reserved	2	

pLruListPrev

is the memory address of the previous cache entry in the LRU list (queue of available buffers).

pLruListNext

is the memory address of the next cache entry in the LRU list.

pDataBuffer

is the memory address of the data buffer for this cache entry.

pHashQueueNext

is the memory address of the next entry in the hash queue.

pHashQueuePrev

is the memory address of the previous entry in the hash queue.

cacheId

is an 8 byte cache identification number (ID) that uniquely identifies the cache entry. The contents of the ID are under the control of the caller.

stausFlags

are 8 bits used as flags to describe status information about this cache entry. The flags are set by the `CacheGetEntry` and `CacheReleaseEntry` operations. The meaning of the bits when set are

Bit	Description
1	Cache entry is sticky.
2	Cache entry is dirty.
3	Cache entry is chained to the previous entry obtained.

Description

The *Cache Pool Descriptor* is the root structure of the cache data structures for a cache pool. It contains the memory addresses of double linked lists threading the cache entries together.

Entries are threaded together in two ways. One set of head and tail pointers links entries containing free (available) data buffers in LRU order. This list is called the *LRU list*. The LRU list contains two types of entries: *invalid* entries for which cache identification numbers (IDs) have not yet been defined are located towards the beginning of the list; *valid* entries with defined IDs are located towards the end.

An entry is also linked to one of several *hash queues*. In the Cache Pool Descriptor, the head and tail of each hash queue is maintained in an array of head and tail address pairs.

(For details on the cacher, see the *CTOS Operating System Concepts Manual*.)

Related Memory Management Structures

Cache Entry Descriptor

This page intentionally left blank.

(continued)

Cache Pool Descriptor

Offset	Field	Size (bytes)	Description
0	signature	4	checks cache pool validity .internally used)
4	pLruListHead	4	memory address of the head of the LRU list
8	pLruListTail	4	memory address of the tail of the LRU list
12	pFlushProc	4	address of flushing procedure or zero
16	flags	1	internally used control flags
17	reserved	1	
18	rgpHashQueues	131*8	doubly linked list of hash queues
1066	cacheStatistics	36	Cache Statistics Block

signature

is a value used internally to verify the validity of the cache pool.

pLruListHead

is the memory address of the first cache entry (head) in the LRU list.

pLruListTail

is the memory address of the last cache entry (tail) in the LRU list.

pFlushProc

is the memory address of the flushing procedure if this cache is a write-back (asynchronous) pool or is zero. (For details on the flushing procedure, see the description of the CacheInit operation in Chapter 3, "Operations.")

flags

are 8 bits used internally as control flags.

rgpHashQueues

is an array of address pairs (8 byte elements) of the heads and tails of each hash queue. The contents of the Cache ID supplied to the CacheGetEntry operation (described in Chapter 3, "Operations") determines which hash queue a cache entry is associated with. A cache entry is in a hash queue only if the entry is valid.

cacheStatistics

is the Cache Statistics Block the contents of which are returned by a call to CacheGetStatistics. (For details, see the description of CacheGetStatistics in Chapter 3, "Operations.")

Description

The *Chords Table* contains information about each chord, including the key post value of the chord, the emulated value of the chord, the identifier of the chord, whether the chord has an LED, and whether the chord remains active until it is pressed again.

This table also identifies whether the chord is exclusionary. If several exclusionary chords are held down at once, only the last one pressed has an effect.

One 10-byte entry exists for each chord. In an emulation data block, a chord table with space for 16 entries is provided.

The Chords Table is a supporting table that resides in both a translation data block and an emulation data block. The *oChords* field of the Translation or Emulation Data Block Header contains the offset, from the beginning of the data block, of the Chords Table.

Related Keyboard Structures

- Allowed States Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Chords Table

Offset	Field	Size (bytes)	Description
0	chordEntry1	10	1st entry in Chords Table
9	chordEntry2	10	2nd entry in Chords Table
⋮			
136	chordEntry16	10	16th entry in Chords Table

chordEntry1

is the 1st entry in the Chords Table. All entries have the following format:

Offset	Field	Size (bytes)	Description
0	emulatedKey	1	In an emulation data block, this value is the emulated keyboard code. In a translation data block, this value is the raw keyboard code.
1	fExclusion	1	A flag. TRUE indicates that the chord has no effect if it is held down at the same time as another exclusionary chord.

Chords Table

(continued)

Offset	Field	Size (bytes)	Description														
2	chordIdentifier	2	<p>is the chord identifier. It is the bit in the operating system's state word that is set when this chord is in effect. The position of the set bit indicates the identity of the chord. By default, each set bit position indicates the following for the K1 keyboard:</p> <table><tr><th>Bit Number</th><th>Chord</th></tr><tr><td>0</td><td>left Shift</td></tr><tr><td>1</td><td>right Shift</td></tr><tr><td>2</td><td>left Code</td></tr><tr><td>3</td><td>right Code</td></tr><tr><td>4</td><td>Lock</td></tr><tr><td>5-15</td><td>Unused</td></tr></table>	Bit Number	Chord	0	left Shift	1	right Shift	2	left Code	3	right Code	4	Lock	5-15	Unused
Bit Number	Chord																
0	left Shift																
1	right Shift																
2	left Code																
3	right Code																
4	Lock																
5-15	Unused																
4	fToggle	1	A flag. TRUE indicates that the chord key remains in effect after it is pressed and released until it is pressed again. FALSE indicates that the chord key no longer remains in effect when it is released.														
5	fLED	1	A flag. TRUE indicates that the chord has an LED.														

(continued)

Chords Table

Offset	Field	Size (bytes)	Description
6	wLEDMask	2	A mask value. The position of the set bit indicates which hardware value must be written to keyboard hardware to turn on/off the chord's LED. (For a description of this word, see the <i>iLed</i> parameter of the SetKbdLed operation.)
8	unemulKey	1	The raw keyboard code of the chord.
9	fInitOn	1	For a toggle chord, TRUE causes the chord to be in effect when the data block is first loaded when a partition is first opened, or when the keyboard is reset.

chordEntry2

is the 2nd entry in the Chords Table.

.
. .
.

chordEntry16

is the 16th entry in the Chords Table.

This page intentionally left blank.

Description

The *Command Masks Table* is used by the keyboard process to identify the translation tables in which a key is defined as a command key. The Command Masks Table is not present unless command keys are used.

Each word entry in the Command Masks Table corresponds to a single key. The first entry corresponds to the key with a keyboard code of 0; the second entry corresponds to the key with a keyboard code of 1, and so forth.

Within an entry, the position of the set bit identifies the translation table in which the key is defined as a command key. As an example, if the first bit (bit 0) in entry 40 is set, a keyboard code of 40 represents a command key if the first translation table is used.

If a key is defined as a command key in the table being used, `ReadKbdInfo` returns a *wStatus* word in which the *command* bit is set. (See `ReadKbdInfo`.)

The Command Masks Table resides in a translation data block. The *cKeys* field of the Translation Data Block Header contains the number of entries in the Command Masks Table. The *oCmdKeyMasks* field of the same header contains the data block offset of the Command Masks Table. If *oCmdKeyMasks* is zero, no Command Masks Table is present.

To create or modify a command masks table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Command Masks Table

Offset	Field*	Size (bytes)	Description
0	commandMask0	2	mask value for keyboard code 0
2	commandMask1	2	mask value for character code 1
.			
.			
510	commandMask255	2	mask value for character code 255

*Command Masks Tables presently contain 256 entries.

commandMask0

is a mask value that identifies the translation table in which a character code of 0 represents command key.

commandMask1

is a mask value that identifies the translation table in which a character code of 1 represents a command key.

.

.

.

.

commandMask255

is a mask value that identifies the translation table in which a character code of 255 represents a command key.

This page intentionally left blank.

Communications Configuration Descriptor

Description

The *Communications Configuration Descriptor* is built by the caller of `AcquireByteStreamC`. The `AcquireByteStreamC` operation is an alternate to `OpenByteStreamC`, allowing the programmer to open the communications byte stream without having a configuration file on disk. Instead, the programmer can define the configuration dynamically through this structure. When `OpenByteStreamC` is called, this structure is built from parameters already set in the communications configuration file.

Related Communications Structures

none

This page intentionally left blank.

(continued)

Communications Configuration Descriptor

Offset	Field	Size (bytes)	Description
0	type	1	is the type of communications device
1	baudRate	2	is the transmit baud rate (0-19200 bps)
3	stopBits	2	the stopBits value is 0-2
5	charSize	1	the charSize value is 5-8
6	parity	1	describes the parity
7	lineControl	1	describes line control mode
8	reserved	2	
10	txMap	1	the output new line mapping modes
11	rxMap	1	the input new line mapping modes
12	txTimeOut	2	time (secs) before a transmission timeout
14	rxTimeOut	2	time (secs) before Device Not Ready
16	fEOF	1	TRUE indicates that bEOF is active.
17	bEOF	1	a character signifying the end of file.
18	sTab	1	the size of the expanded tab
19	cCharsPerLine	1	the number of characters per line
20	sbTxltnFile (79)	1	the translation file name
99	fNRZI	1	TRUE is Non Return to Zero Inverted mode
100	rxBaudRate	2	the receive baud rate (0-19200 bps).

type

is the communications device type. The types and their corresponding values are

Value	Type
1	Parallel Printer

Communications Configuration Descriptor

(continued)

Value	Type
2	Serial Printer
3	Communications

baudRate

is the transmit baud rate (0-19200 bits per second).

stopBits

is the stopBits value. This value is in the range of 0 to 2.

charSize

Is the charSize value. The value is in the range of 5 to 8.

parity

is the parity status. The parity values and their meanings are:

Value	Meaning
0	No Parity
1	Even Parity
2	Odd Parity
3	One Parity
4	Zero Parity

lineControl

is the line control mode. The modes and their values are:

Value	Mode
0	No Line Control

(continued)

Communications Configuration Descriptor

Value	Mode
1	XON/XOFF Mode
2	Clear to Send (CTS) Mode
3	Both XON/XOFF and CTS Modes

txMap

is the output new line mapping mode. The output mapping modes and their values are:

Value	Mode	Translation
0	Binary Mapping	New line to new line Carriage return to carriage return
1	New Line	New line to carriage return
1	Carriage Return	Carriage return to carriage return
2	Carriage Return/ Line Feed	New line to carriage return plus new line Carriage return to carriage return

rxMap

is the input new line mapping mode. The input mapping modes and their values are:

Value	Mode	Translation
0	Binary Mapping	New line to new line Carriage return to carriage return
1	New Line	New line to new line
1	Carriage Return	Carriage return to new line
2	Carriage Return/ Line Feed	New line to new line Carriage return to new line

txTimeOut

is the time in seconds before a timeout on a transmission.

rxTimeOut

is the time in seconds if no characters are received before a status code indicating "Device not ready" is returned.

fEOF

is a flag. TRUE indicates that bEOF is active.

(continued)

Communications Configuration Descriptor

bEOF

is a character signifying the end of file.

sTab

is the size (number of spaces) of the expanded tab.

cCharsPerLine

is the number of bytes after which a new line mode is generated (based on TxMap).

sbTxlnFile (79)

contains the translation file name (string). The first byte of the string is the string length.

fNRZI

is a flag. TRUE indicates Non Return to Zero Inverted mode (for XC-002 hardware only).

rxBaudRate

is the receive baud rate (0 to 19200 bits per second).

This page intentionally left blank.

Description

The *communications status buffer* contains information about cluster line activity. The **Cluster Status** command calls the `GetClusterStatus` operation to access this structure. For details on the information returned by **Cluster Status**, see the *CTOS System Administration Guide*.

Depending on its parameters, the communications status buffer returns either server information or workstation information. For each workstation user, the workstation `SignOn` user name and the contents of the Drop Control Table (dct) are returned. A portion of the Line Control Block is returned for the server.

Related Cluster Structures

none

This page intentionally left blank

(continued)

Communications Status Buffer

Offset	Field	Size (bytes)	Description
Workstation Information			
0	rgbWsName	31	SignOn user name for this workstation
32	next	2	offset of the next DCT
34	prev	2	offset of the previous DCT
36	station	1	the polled ID of this workstation
37	lineState	1	internal state of DCT
38	saXBlockOut	2	address of queue of outstanding requests
40	saXBlockIn	2	address of queue of incoming responses
42	nOutstandingRq	1	outstanding requests using large X-Blocks
43	nOutstandingSmallRq	1	outstanding requests using small X-blocks
44	pollSequenceNumber	4	number of polls of this workstation
48	erc	2	last error recognized on the cluster line
50	NR	1	IFrame sequence number received
51	NS	1	IFrame sequence number sent
52	fh	2	file handle for booting or dumping
54	lfa	4	logical file address for booting or dumping
58	revisionLevel	1	workstation internal version string
59	osType	1	workstation operating system type
60	actionCode	1	state of the workstation (booting, dumping)
61	fActivePollCycle	1	TRUE if this is the workstaion being polled
62	nCrcError	2	number of cyclic redundancy check errors
64	nOverrun	2	number of overrun errors
66	nSequenceError	2	number of sequence errors
68	nProtocolError	2	number of protocol errors
70	nAddressError	2	number of address errors
72	nLengthError	2	number of length errors
74	nTimeout	2	number of timeout errors
76	wsnCrcError	2	number of cyclic redundancy check errors
78	wsnOverrun	2	number of overrun errors
80	wsnSequenceError	2	number of sequence errors
82	wsnProtocolError	2	number of protocol errors
(continued)			

Workstation Information

84	wsnAddressError	2	number of address errors
86	wsnLengthError	2	number of length errors
88	wsnTimeout	2	number of timeout errors
90	dctWsSumSimpleRqTime	4	time the request took to be processed
94	dctnSimpleRq	4	number of requests processed by server
98	dctWsMaxSimpleRqTime	2	maximum time to process a request
100	dctWsSumGetDtRqTime	4	time to process a GetDateTime request
104	dctnGetDtRq	2	number of GetDateTime requests
106	dctWsMaxGetDtRqTime	2	max time to process GetDateTime request
108	dctWsSumBlockRqTime	4	time to process a piecemeal request
112	dctWsnBlockRq	4	number of piecemeal requests
116	dctWsMaxBlockRqTime	2	max time to process a piecemeal request
118	dctWsnTicksPerSecond	2	number of clock ticks per second
120	nIFrames	4	number of IFrames sent to server
124	nPagesRead	4	number of blocks read
128	nPagesWritten	4	number of blocks written
130	nTicksSinceLastPoll	2	number of ticks since last poll
132	nTicksMax	2	maximum number of ticks between polls
134	dctPollSysTime	4	counter to time the polls
138	nTimesPolled	4	times this workstation was polled
142	lastActivePollCycle	2	not currently used

Server Workstation Information

0	revisionLevel	2	server version string
2	nWsActive	2	number of active workstations
4	nWsTotal	2	total number of configured workstations
6	nXBlocksFree	1	number of free large X-blocks
7	nXBlocksSmallFree	1	number of free small X-blocks
8	nCrcError	2	number of cyclic redundancy check errors
10	nOverrunError	2	number of overrun errors
12	nSequenceError	2	number of sequence errors
14	nProtocolError	2	number of protocol errors
16	nAddressError	2	number of address errors
18	nLengthError	2	number of length errors
20	nTimeout	2	number of timeout errors
22	nWsDownTimeout	2	number of workstation timeouts
24	nWsDownErrors	2	number of disconnected workstations

(continued)

(continued)

Communications Status Buffer

Server Workstation Information

26	nWsBootRequest	2	workstations booted from server
28	nWsBootComplete	2	workstations that completed boot
30	nWsAccessLinkRequest	2	booted workstations in contact with server
32	nWsDumpRequest	2	workstations that dumped to the server
34	nWsDumpComplete	2	workstations completing dump to server
36	maxTicksBetweenPolls	2	maximum number of ticks between polls
38	nSnrm	4	number of SNRMs sent out
42	nSnrmReply	4	(internal) number of SNRMs responded to
46	nSnrmErrorReply	4	errors received on reply to a SNRM
50	nFalseTimeout	2	(internal) count of false timeouts
52	nXBlockWaits	4	number of times large X-Block not available
56	nXBlockSmallWaits	4	times small X-Block were not available
60	nIFramesRrIn	4	RR IFrame requests that came in
64	nIFramesRnrIn	4	RNR IFrame requests that came in
68	nIFramesRrOut	4	number of RR IFrame requests sent out
72	nIFramesRnrOut	4	number of RNR IFrame requests sent out
76	nPagesRead	4	blocks read by each workstation
80	nPagesWritten	4	blocks written by each workstation
84	statIdleTicksLastSec	2	number of idle ticks since the last second
86	statIdleTicksLast10Sec	2	number of idle ticks in the last 10 seconds
88	statsSeconds	4	time length for logging cluster line statistics
92	pollSequenceNumber	4	running count of server polls
96	nRnrIn	4	server polls with RNR requests coming in
100	nRrIn	4	server polls with RR requests coming in
104	nRnrOut	4	server polls with RNR requests sent out
108	nRrOut	4	server polls with RR requests sent out
112	statsfHighSpeed	1	cluster line speed
113	statsSbVerRun	31	server version string
144	ticksPerSecond	2	number of ticks in the last second
146	ticksSinceLastPoll	2	number of ticks since the last poll
148	lcbPollSysTime	4	counter to time polls
152	nXBlocksTotal	2	total number of large X-Blocks allocated
154	nXBlocksSmallTotal	2	total number of small X-Blocks allocated
156	saPerXBlock	2	offset to size of the large X-Blocks
158	saPerSmallRq	2	offset to size of the small X-Blocks
160	nXBlockBoundary	2	large X-Blocks crossing boundaries
162	nXBlockSmallBoundary	2	small X-Blocks crossing boundaries

(continued)

Server Workstation Information

164	fCountersReset	1	flag to reset counters at midnight
165	lastSnrmSent	1	ID number of the last SNRM sent
166	ppMeterBuffer	4	location of stored lframe information
170	pMasterStats	4	reserved
174	oRgDct	4	reserved
176	reserved	4	not used
180	nYBlksMax	4	highest number of free Y blocks at a time
184	nYBlksMin	4	lowest number of free Y blocks at a time
188	nYBlksAvail	4	number of available Y blocks
192	nZBlksMax	4	highest number of free Z blocks at a time
196	nZBlksMin	4	lowest number of free Z blocks at a time
200	nZBlksAvail	4	number of available Z blocks
204	n4ByteRnr	4	counter of 4-byte RNR frames
208	fResetCounters	2	flag set so counters not reset at midnight
210	nRcvError	4	number of RCV frames that caused errors
214	nUnderrun	4	number of underruns

Workstation Information

rgbWsName

is SignOn user name for this workstation.

next

is the offset to the next DCT.

prev

is the offset to the previous DCT.

station

is the polled identification number (ID) of this workstation. This ID value can change if the server is rebooted.

lineState

is the internal state of DCT (such as whether it is booting or dumping).

saXBlockOut

is the memory address of the queue of outstanding requests.

saXBlockIn

is the memory address of the queue of incoming responses.

nOutstandingRq

is the number of outstanding requests that fit in large X-blocks for this workstation. (X-block sizes are shown in the **Cluster Status** command.)

nOutstandingSmallRq

is the number of outstanding requests for this workstation.

pollSequenceNumber

is the number of times this workstation was polled.

erc

is the last error recognized on the cluster line.

NR

is the sequence number (0 to 7) of information frames (IFrames) received by this workstation.

NS

is the sequence number (0 to 7) of IFrames sent by this workstation to the server.

fh

is the file handle used for bootstrapping or dumping.

lfa

is the logical file address used for bootstrapping or dumping.

revisionLevel

is an internal version string for the workstation. The server reads this string to determine what code to use when communicating with this workstation.

osType

indicates the type of workstation operating system (such as 252) to bootstrap or dump.

actionCode

is the state of the server (for example, booting, dumping, or polling a workstation).

fActivePollCycle

is a flag that is TRUE if this workstation is the workstation currently being polled.

nCrcError

is the number of cyclic redundancy check (CRC) errors detected at the server. A CRC error occurs if the CRC at the end of the frame does not match the CRC calculated by the chip when the frame was received. (For information on CRCs, see the *CTOS System Administration Guide*.)

nOverrun

is the number of overrun errors detected at the server. (See the *CTOS System Administration Guide*.)

nSequenceError

is the number of sequence errors detected at the server. (See the *CTOS System Administration Guide*.)

nProtocolError

is the number of protocol errors detected at the server. (See the *CTOS System Administration Guide*.)

nAddressError

is the number of address errors detected at the server. (See the *CTOS System Administration Guide*.)

nLengthError

is the number of length errors detected at the server. (See the *CTOS System Administration Guide*.)

nTimeout

is the number of timeout errors detected at the server. (See the *CTOS System Administration Guide*.)

wsnCrcError

See *nCrcError*. This field applies to workstation errors.

wsnOverrun

See *nOverrun*. This field applies to workstation errors.

wsnSequenceError

See *nSequenceError*. This field applies to workstation errors.

wsnProtocolError

See *nProtocolError*. This field applies to workstation errors.

wsnAddressError

See *nAddressError*. This field applies to workstation errors.

wsnLengthError

See *nLengthError*. This field applies to workstation errors.

wsnTimeout

See *nTimeout*. This field applies to workstation errors.

dctWsSumSimpleRqTime

time request took to be processed.

dctnSimpleRq

is the number of requests processed by the server.

dctWsMaxSimpleRqTime

is the maximum time to process a request.

dctWsSumGetDtRqTime

is the time to process GetDateTime requests.

dctnGetDtRq

is the number of GetDateTime requests.

dctWsMaxGetDtRqTime

is the maximum time to process GetDateTime requests.

dctWsSumBlockRqTime

is the time to process piecemeal requests.

dctWsnBlockRq

is the number of piecemeal requests.

dctWsMaxBlockRqTime

is the maximum time to process piecemeal requests.

dctWsnTicksPerSecond

is the number of clock ticks per second.

nIFrames

is the number of IFrames sent to the server.

nPagesRead

is the number of blocks read by this workstation.

nPagesWritten

is the number of blocks written by this workstation.

nTicksSinceLastPoll

is number of ticks since the last poll of this workstation. This value is used to determine the number of seconds since the last poll in the **Cluster Status** command time menu (function key F7).

nTicksMax

is maximum number of ticks between polls. This is used to determine the the maximum poll interval in the **Cluster Status** command time menu.

dctPollSysTime

is a counter to time the polls.

nTimesPolled

is the number of times this workstation was polled.

Server Workstation Information

revisionLevel

is the server version string.

nWsActive

is the number of active workstations.

nWsTotal

is the total number of configured workstations.

nXBlocksFree

is the number of free large X-blocks. See the blocks menu (function key F7) in the **Cluster Status** command.

nXBlocksSmallFree

is the number of free small X-blocks. See the blocks menu in the **Cluster Status** command.

nCrcError

is the number of cyclic redundancy check (CRC) errors detected at the server. A CRC error occurs if the CRC at the end of the frame does not match the CRC calculated by the chip when the frame was received. (For information on CRCs, see the *CTOS System Administration Guide*.)

nOverrunError

is the number of overrun errors detected at the server. (See the *CTOS System Administration Guide*.)

nSequenceError

is the number of sequence errors detected at the server. (See the *CTOS System Administration Guide*.)

nProtocolError

is the number of protocol errors detected at the server. (See the *CTOS System Administration Guide*.)

nAddressError

is the number of address errors detected at the server. (See the *CTOS System Administration Guide*.)

nLengthError

is the number of length errors detected at the server. (See the *CTOS System Administration Guide*.)

nTimeout

is the number of timeout errors detected at the server. (See the *CTOS System Administration Guide*.)

nWsDownTimeout

is the number of workstation timeouts.

nWsDownErrors

is the number of workstations that are disconnected from the server.

nWsBootRequest

is the number of workstations that were bootstrapped from the server.

nWsBootComplete

is the number of workstations that completed bootstrapping from the server.

nWsAccessLinkRequest

is the number of workstations that were bootstrapped from the server and established contact.

nWsDumpRequest

is the number of workstations that dumped to the server.

nWsDumpComplete

is the number of workstations that completed a dump to the server.

maxTicksBetweenPolls

is maximum number of ticks between polls. This is used to determine the maximum poll interval in the **Cluster Status** command time menu.

nSnrm

specifies the number of Set Normal Response Modes (SNRMs) sent out. See the the **Cluster Status** command frames menu (function key F6).

nSnrmReply

is an internal use value indicating the number of SNRMs responded to.

nSnrmErrorReply

is an internal use value indicating the number of times the server received an error on a reply to a SNRM.

nFalseTimeout

is an internal count of false timeouts.

nXBlockWaits

is the number of times the server needed a large X-Block when none were available.

nXBlockSmallWaits

is the number of times the server needed a small X-Block when none were available.

nIFramesRrIn

is the number of Receiver Ready IFrame requests that came in from workstations. (See the **Cluster Status** frames menu.) RR means that the requestor currently does have a buffer available for the information requested.

nIFramesRnrIn

is the number of Receiver Not Ready (RNR) IFrame requests that came in from workstations. (See the **Cluster Status** frames menu.) RNR means that the requestor currently does not have a buffer available for the information requested.

nIFramesRrOut

is the number of Receiver Ready IFrame requests that were sent out to workstations. (See the **Cluster Status** frames menu.)

nIFramesRnrOut

is the number of Receiver Not Ready (RNR) IFrame requests that went out to workstations. (See the **Cluster Status** frames menu.)

nPagesRead

is the number of blocks read by each workstation. See the **Cluster Status** data menu (function key **F4**).

nPagesWritten

is the number of blocks written by each workstation. See the **Cluster Status** data menu.

statIdleTicksLastSec

is the number of idle ticks during which there was no line activity since the last second. This value is used to determine the percentage of line use in the last second in the **Cluster Status** command header.

statIdleTicksLast10Sec

is the number of idle ticks during which there was no line activity in the last 10 seconds. This value is used to determine the percentage of line use in the last 10 seconds in the **Cluster Status** command header.

statsSeconds

is the time period during which the statistics values in the current **Cluster Status** display have been logged.

pollSequenceNumber

is the running count of the number of times the server has polled since statistics started being logged.

nRnrIn

is the number of times the server has polled, and RNR IFrame requests have come in from workstations.

nRnrOut

is the number of times the server has polled, and RNR IFrame requests were sent out to workstations.

statsfHighSpeed

is the cluster line speed.

statsSbVerRun

is the operating system version string passed to the **Cluster Status** command.

ticksPerSecond

is the number of ticks in the last second. This value is used to determine the number of timeouts.

ticksSinceLastPoll

is the number of ticks since the last poll. This value is used to determine the number of timeouts.

lcbPollSysTime

is the counter to time polls.

nXBlocksTotal

is the total number of large X-Blocks allocated.

nXBlocksSmallTot

is the total number of small X-Blocks allocated.

saPerXBlock

is the offset to a value indicating the size of the large X-Blocks.

saPerSmallRq

is the offset to a value indicating the size of the small X-Blocks.

nXBlockBoundary

specifies the number of large X-Blocks that cross physical boundaries so that they cannot be used for Direct Memory Access (DMA).

nXBlockSmallBoundary

specifies the number of small X-Blocks that cross physical boundaries so that they cannot be used for Direct Memory Access (DMA).

fCountersReset

is a flag. TRUE resets the counters at midnight.

lastSnrmSent

is the identification number of the last SNRM sent.

ppMeterBuffer

is the memory address of a pointer to a memory area where Iframe information is stored for debugging and tuning the cluster. This information is accessed using function key F1 (meter) in the **Cluster Status** command. (See the *CTOS System Administration Guide*.)

pMasterStats

is reserved.

oRgDct

is reserved.

nYBlksMax

is the maximum number of Y blocks that have been available at one time.

nYBlksMin

is the lowest number of Y blocks that have been available at one time.

nYBlksAvail

is the number of Y blocks currently available.

nZBlksMax

is the highest number of free Z blocks that have been available at one time.

(continued)

Communications Status Buffer

nZBlksMin

is the lowest number of free Z blocks that have been available at one time.

nZBlksAvail

is the number of Z blocks currently available.

n4ByteRnr

is the counter of 4-byte RNR IFrames.

fResetCounters

is a flag. If TRUE, the counters are not reset at midnight.

nRcvError

is the counter of RCV frames that caused errors.

nUnderrun

is the number of underruns (protocol errors as a result of hardware problems).

This page intentionally left blank

Description

The *Conditions Table* determines the emulation or translation table that the keyboard process uses when a particular combination of chord keys is pressed or toggled.

The Conditions Table is a support table that resides in both an emulation data block and a translation data block. The data block offset of the Conditions Table is located in the *oConditions* field of the Translation or Emulation Data Block Header. The count of Conditions Table entries is located in the *cConditions* field of either header. A maximum of 16 different translation or emulation tables may be specified in a Conditions Table.

With the Keyboard Customization Tool, the end user can construct or modify a conditions table.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Conditions Table

Offset	Field*	Size (bytes)	Description
0	condition1	8	first entry in the Conditions Table
8	condition2	8	second entry in the Conditions Table
.			
.			
.			

*Conditions Tables contain a variable number of entries.

condition1

is the first entry in the Conditions Table.

condition2

is the second entry in the Conditions Table.

.

.

.

and so forth.

This page intentionally left blank.

Description

The *Control Chords Table* is used by the keyboard process to identify chord keys that perform similar functions.

In this table, each entry contains the identifiers of chords that fall into a particular category. If the identifier of the pressed chord key is present in the entry, the chord key is recognized as a specific type of chord. For example, if the identifier of the chord is present in the *allShiftsMask* field, the chord is recognized as a **Shift** key.

The Control Chords Table is a support table that resides in a translation data block. The data block offset of the Control Chords Table is located in the *oControlChords* field of the Translation Data Block Header. The count of entries in the Control Chords Table is located in the *cControlChords* field of the same Header.

With the Keyboard Customization Tool, the user can construct or modify a Control Chords Table.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Control Chords Table

Offset	Field	Size (bytes)	Description
0	allShiftsMask	2	value that tests for Shift keys
2	bullShiftMask	2	value that tests for Bull Shift keys
4	noDiacritMask	2	value that tests for non-diacritic keys
6	noRepeatMask	2	value that tests for repeat-inhibiting keys
8	codeKeyMask	2	value that tests for Code keys
10	altCodeKeyMask	2	value that tests for Alt and Code keys
12	k1ShiftMask	2	value that tests for K1 Shift keys
14	lockKeyMask	2	value that tests for Lock key
16	reserved		reserved for operating system use
18	reserved		reserved for operating system use
20	reserved		reserved for operating system use

allShiftsMask

is a value that tests for **Shift** keys. For this entry as well as subsequent entries, the default is 0.

bullShiftMask

is a value that tests for **Shift** keys on the Bull keyboard.

noDiacritMask

is a value that tests for keys that prevent a diacritic pair from being formed.

noRepeatMask

is a value that tests for keys that prevent other keys from repeating.

codeKeyMask

is a value that tests for **Code** keys.

altCodeKeyMask

is a value that tests for **Alt** and **Code** keys.

Control Chords Table

(continued)

k1ShiftMask

is a value that tests for **Shift** keys on the K1 keyboard.

lockKeyMask

is a value that tests for the **Lock** key.

Description

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Each processor board in a shared resource processor contains a *CPU description table (CDT)*. The CDT described next is supported by CTOS/XE 3.0 and higher. For comparison purposes, the earlier version of the CDT (supporting systems such as Shared Resource Processor, Version 1.4) is referred to as the pre-3.0 CDT. [For a detailed description of the fields in the pre-3.0 CDT, see "CPU Description Table (pre-3.0)," later in this chapter.]

The CDT for each processor describes that processor to other processors. To accomplish this, the CDT contains configuration, routing, and initialization information as well as a pointer to an area reserved for inter-CPU communication (ICC) buffers.

With CTOS/XE Version 3.0, the following changes affected fields in the CDT. Space left over as a result of obsoleted fields is reserved for future use.

1. The method of queuing foreign board requests and responses doesn't use circular buffers within the CDT. As a result, the following pre-3.0 CDT request/response circular buffer fields are obsolete:

Offset	Field
10	ibRqTakePtr
12	ibRqPutPtr
14	ibRqStartPtr
16	ibRqEndPtr
18	ibRespTakePtr
20	ibRespPutPtr
22	iRespStartPtr
24	iRespEndPtr

2. Furthermore, with CTOS/XE 3.0 and higher, you cannot use the CdtIO facility for communicating with a processor board. You must either use Clusterview (remote keyboard/video service) or hook up a terminal directly to the board with which you want to communicate. As a result, the following pre-3.0 CDT fields for CdtIO functions are obsolete:

Offset	Field
35	fCdtIO
290	bTTIStatus
291	bTTIData
292	bTTOSStatus
293	bTTOData

3. In addition, because each processor manages disabling and enabling of its own cluster lines, the following fields are obsolete:

Offset	Field
299	fDisCls
300	wCpWho
302	wCpTimer

4. The following pre-3.0 CDT fields in the master processor are obsolete because they are not used:

Offset	Field
388	oTermTable
390	sTermTable

(continued)

CPU Description Table

5. Functionality of the following pre-3.0 CDT fields in the master processor has been replaced by the requests, GetRouteTable and UpdateRouteTable:

Offset	Field
40	rgbFpXlate
392	oLineTable
394	sLineTable
396	oRqRoute
430	sRqRoute

6. The CDT for 3.0 introduces the following new fields:

Offset	Field
0	paIccSeg
10	fWashDisable
22	qMemorySize
294	wPanelStatus
296	bFrontPanel
297	fDegraded
298	bDumpCommand
304	verifyCode
305	bBoardsNotReady
306	paConfigFile
310	cbConfigFile
312	cLocalUserNums
314	userNumMax
376h	oHandleTable
378	sHandleTable
380	oHandleFree
382	nDevMounted
384	paMasterMountTable
388	rgObit

You can obtain the address of the CDT by calling GetPStructure with a *structCode* value of 27.

Related Shared Resource Processor Structures

CPU Description Table (pre-3.0)

Terminal Output Buffer

(continued)

CPU Description Table

Offset	Field	Size (bytes)	Description
0	palccSeg	4	physical address of ICC data
4	reserved	4	
8	bHardwareType	1	describes the hardware type
9	fWatchDog	1	flag set once/sec by the master processor
10	fWashDisable	1	used for hardware debugging
11	reserved	11	
22	qMemorySize	4	memory size in K-bytes/128
26	fLockByte	1	ensures no request/response buffer race
27	bInitErrorStatus	1	error detection at initialization placed here
28	bMemorySize	1	memory size in K-bytes/128
29	bBootStruct\$FF	1	signature, value = 0FFh
30	bBootStruct\$0	1	signature, value = 0
31	bBootStruct\$A6	1	signature, value = 0A6h
32	bBootCommand	1	commands to bootstrap
33	bMasterFP	1	master processor slot number
34	fOsInitialized	1	TRUE when OS has initialized
35	reserved	13	
48	rgbBusConfigTable	240	array of indexed slot numbers
288	nUserNumber	2	first user number for this board
290	reserved	4	
294	wPanelStatus	2	front panel key switch position
296	bFrontPanel	1	hex value of front panel LED
297	fDegraded	1	booted normal/remote and bus repeater off
298	bDumpCommand	1	indicates bootstrap or dump procedure
299	reserved	5	
304	verifyCode	1	number of times server was rebooted
305	bBoardsNotReady	1	count used to synchronize boards at init
306	paConfigFile	4	physical address of configuration file
310	cbConfigFile	2	byte count of configuration file

(continued)

CPU Description Table

(continued)

Offset	Field	Size (bytes)	Description
312	cLocalUserNums	2	maximum user numbers for this board
314	userNumMax	2	largest user number on this board
316	reserved	60	
376	oHandleTable	2	list of global handles for rDevice requests
378	sHandleTable	2	size of global handle list area
380	oHandleFree	2	offset of next free global handle
382	nDevMounted	2	number of devices in FP mount table
384	paMasterMountTable	4	physical address of FP mount table
388	rgObit	5	array of processor status bytes
393	reserved	3	

paIccSeg

is the physical address of segment used for ICC by this processor board. The fields in the ICC segment are described below. (See "Inter-CPU Communication" in the *CTOS Operating System Concepts Manual* for details on ICC.)

Offset	Field	Size (bytes)	Description
0	listFreeZ	32	is the ring queue of free Z-blocks.
32	listFreeY	32	is the ring queue of free Y-blocks.
64	listFreeW	32	is the ring queue of free W-blocks.
96	listRequest	16	is the ring queue of incoming requests (already in a buffer in local memory on this board).
112	listRespond	16	is the ring queue of incoming responses.
128	orgPaBuffer	2	is an array of bus addresses of all the block type buffers contained within this ICC segment.

(continued)

CPU Description Table

Offset	Field	Size (bytes)	Description
130	<i>orgOwnBuffer</i>	2	is the table indicating which processor is using a buffer for ICC. The table is used for recovery from a crashed board to identify and free any outstanding buffers.
132	<i>mpiSlotfWaitSatisfied</i>	4	is the table of flags indicating which processors are waiting to use a block. (See "Inter-CPU Communication" in the <i>CTOS Operating System Concepts Manual</i> .)
136	<i>rgObit</i>	37	is an array of 1 byte processor status codes, with one byte corresponding to each possible processor on a shared resource processor .
173	signature	3	identifies a valid processor as opposed to a processor that crashed or failed to boot.
176	<i>mpcParcRq</i>	1024	is a block size Histogram that analyzes request block sizes of incoming requests to a board. This analysis allows block sizes to be customized. <i>mpcParcRq</i> maps the count of paragraphs to the count of requests of that size.

bHardwareType

is the hardware type. The hardware types and their values are shown below. The hardware type for the local processor can also be obtained through the *hardwareType* field in the System Configuration Block.

Value	Type
10	File Processor
11	Terminal Processor
12	Cluster Processor
13	Storage Processor
14	Data Processor

Value	Type
19	Boot Tape (booted when <i>fOsLoader</i> is TRUE. See "Operating System Flags" in this chapter.)
20	General Processor
21	GP + communications interface
22	GP + SCSI interface
64	Obsolete field (Applications Processor)
65	Obsolete field (Applications Processor2)

fWatchDog

is a flag that the master processor sets to TRUE once per second for each processor board. *fWatchDog* must be cleared within one second by the processor board or the board is considered dead.

fWashDisable

is a flag that disables reading of memory locations by the Error Code Correction (ECC) washer, for example, when hardware debugging tools are being used.

qMemorySize

is the memory size in bytes.

fLockByte

is used to ensure no race for the request and response buffers.

bInitErrorStatus

is the byte at which the code for an error detected at initialization is placed.

bMemorySize

is memory size in K-bytes/128. This is set by the bootstrap ROM.

bBootStruct\$FF

signature, value = 0FFh.

bBootStruct\$0

signature, value = 0.

bBootStruct\$A6

signature, value = 0A6h.

bBootCommand

is a command/status byte that is set by the boot ROM. The values are

Value	Command/Status
0	fRunning
1	fBootMe
2	fDumpThenBoot
3	fDumpThenError
240	fFailedDiagnostics
241	fRunningDiagnostics

bDumpCommand

is a command/status byte that is set by the master processor when this board is bootstrapped. The initial values are set in a shared resource processor configuration file. (See the *CTOS System Administration Guide* for details.) The values are

Value	Command/Status
0	BootStrap only
1	Dump low memory
2	Dump all of memory

bMasterFp

is the master processor slot number.

fOsInitialized

is a flag that is TRUE when the operating system has initialized.

rgbBusConfigTable

is an array of indexed slot numbers.

nUserNumber

is the first in the series of user numbers for this processor board. *cLocalUserNums* (at offset 312) is the total number of user numbers for this board.

wPanelStatus

is the front panel key switch position. The values of the keyswitch positions are:

Value	Key switch position
0	manual
4	remote
16	normal

bFrontPanel

is the hexadecimal value of the front panel LED.

fDegraded

is a flag that is TRUE if the shared resource processor was booted in the normal or remote key switch position and a bus repeater was off.

verifyCode

is the number of times the server has been rebooted. This number recycles to 1. It is never 0 except on the master processor.

paConfigFile

is the physical address of the configuration file. This address is valid only during the time the board is being booted. After initialization, the address is no longer valid.

cbConfigFile

is the count of bytes in the configuration file.

bBoardsNotReady

is a counter used by the master processor to synchronize boards at initialization time. As each board is bootstrapped, the master processor increments the count. With each booted board that completes initialization, the count is decremented.

cLocalUserNums

is the maximum number of user numbers allowed on this processor board. This number is the value of *nPartitions* in the file *SysGen.asm* and does not include cluster or network user numbers.

oHandleTable

is the offset the list of global handles for requests routed by the shared resource processor routing type *rDevice* (master processor only). (For details on routing by handle, see "Interprocess Communication" in the *CTOS Operating System Concepts Manual*. The shared resource processor routing types are described in "Inter-CPU Communication" in that same manual.)

sHandleTable

is the size in bytes of the memory area containing the list of global handles (master processor only).

oHandleFree

is the offset of the next free global handle (master processor only).

nDevMounted

is the number of devices in the master processor mount table (master processor only).

paMasterMountTable

is the physical address of the master processor mount table (master processor only).

rgObit

is an array of 1 byte processor status codes, with one byte corresponding to each possible processor on a shared resource processor.

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

Each processor board in a shared resource processor contains a *CPU description table (CDT)*. The CDT described next is for the 1.4 and earlier shared resource processor versions. If your shared resource processor is CTOS/XE 3.0 or higher, you will need to refer to "CPU Description Table," earlier in this chapter.

The CDT (pre-3.0) describes that processor to other processors. To accomplish this, the CDT contains offsets of circular buffers used by other processors to send inter-CPU Communication (ICC) requests and responses, and contains some routing information. The master processor CDT contains additional routing information and tables used to translate line and terminal numbers into particular slot number-port number pairs. (See the descriptions of offsets 388 through 430.)

You can obtain the address of the CDT by calling `GetPStructure` with a *structCode* value of 27.

Related Shared Resource Processor Structures

- CPU Description Table
- Terminal Output Buffer

This page intentionally left blank

(continued)

CPU Description Table (pre-3.0)

Offset	Field	Size (bytes)	Description
0	reserved	8	
8	bHardwareType	1	describes the type of processor
9	fWatchDog	1	flag set every second by master processor
10	ibRqTakePtr	2	current request linear offset
12	ibRqPutPtr	2	next available request's linear offset
14	ibRqStartPtr	2	base of request circular buffer
16	ibRqEndPtr	2	end of request circular buffer
18	ibRespTakePtr	2	current response linear offset
20	ibRespPutPtr	2	next available response's linear offset
22	ibRespStarPtr	2	base of response circular buffer
24	ibRespEndPtr	2	end of response circular buffer
26	fLockByte	1	ensures no race for circular buffers above
27	blnitErrorStat	1	error detection at initialization placed here
28	bMemorySize	1	memory size in K-bytes/128
29	bBootStruct\$FF	1	signature, value = 0FFh
30	bBootStruct\$0	1	signature, value = 0
31	bBootStruct\$A6	1	signature, value = 0A6h
32	bBootCommand	1	commands to bootstrap
33	bMasterFp	1	master processor slot number
34	fOsInitialized	1	TRUE when OS has initialized
35	fCdtIO	1	communication is by CdtIO facility
36	reserved	4	
40	rgbFPXlate	8	array of FP slot numbers
48	rgbBusConfigTable	240	array of indexed slot numbers
288	reserved	2	
290	bTTIStatus	1	keystroke is available to kbd process
291	bTTIData	1	buffer into which keystroke is placed
292	bTTOStatus	1	display character available to terminal
293	bTTOData	1	buffer into which character is placed
294	reserved	94	
388	oTermTable	2	translates terminal to slot number/port
390	sTermTable	2	size of preceding table
392	oLineTable	2	translates line number to slot number/port
394	sLineTable	2	size of preceding table
396	oRqRoute (17)	34	routing information per request level
430	sRqRoute (17)	34	size of each level

bHardwareType

same description as *bHardwareType* in "CPU Description Table," earlier in this chapter.

fWatchDog

same as *fWatchDog* in "CPU Description Table."

ibRqTakePtr

is the current request linear offset.

ibRqPutPtr

is the next available request's linear offset.

ibRqStartPtr

is the base of request circular buffer.

ibRqEndPtr

is the end of request circular buffer

ibRespTakePtr

is the current response linear offset.

ibRespPutPtr

is the next available response's linear offset.

ibRespStartPtr

is the base of the response circular buffer.

ibRespEndPtr

is the end of the response circular buffer.

fLockByte

same as *fLockByte* in "CPU Description Table."

bInitErrorStat

same as *bInitErrorStat* in "CPU Description Table."

bMemorySize

same as *bMemorySize* in "CPU Description Table."

bBootStruct\$FF

see "CPU Description Table."

bBootStruct\$0

see "CPU Description Table."

bBootStruct\$A6

see "CPU Description Table."

bBootCommand

see "CPU Description Table."

bMasterFp

see "CPU Description Table."

fOsInitialized

see "CPU Description Table."

fCdtIO

is a flag that is TRUE if communication with this board is by CdtIO facility.

rgbFPXlate

is an array of FP slot numbers.

rgbBusConfigTable

see "CPU Description Table."

bTTIStatus

is used by the CdtIO facility. A value of TRUE means a keystroke (byte) is available to the keyboard polling process.

bTTIData

is a 1-byte buffer into which the keystroke value available to the keyboard polling process is placed.

bTTOStatus

is used by the CdtIO facility. A value of TRUE means a display character (byte) is available to the terminal video device.

bTTOData

is a 1-byte buffer into which the display character available to the terminal is placed.

oTermTable

is a linear offset of table that translates the terminal number to slot number/port (master processor only).

sTermTable

is the size of *oTermTable* (master processor only).

oLineTable

is a linear offset of table that translates the line number to slot number/port (master processor only).

sLineTable

is the size of *oLineTable* (master processor only).

oRqRoute (17)

is an array of routing information kept on a per-request level basis (master processor only).

sRqRoute (17)

is the size of each request level (master processor only).

This page intentionally left blank

Description

The *Data Control Structure* is used by the Telephone Service data management facility. Data management transfers data over telephone lines between modems in Voice Processor modules. The Data Control Structure is passed to the Telephone Service with the `TsDataOpenLine` operation to control the data call characteristics such as baud rate, parity, or line control. The `TsDataRetrieveParams` and `TsDataChangeParams` operations can be used to query or change the parameters of the specified line.

The Voice/Data services operations are described in Chapter 3, "Operations." For details on how to write voice response applications to be used with Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice Control Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued)

Data Control Structure

Offset	Field	Size (bytes)	Description
0	openMode	1	data open mode
1	fOriginate	1	TRUE programs modem in originate mode
2	cTimeout	2	maximum time to complete the open
4	baudrate	2	either 300 or 1200
6	parity	1	is the parity control
7	lineControl	1	sets flow control mode
8	fLL	1	termination request doesn't close service
9	fEndOfBlock	1	<i>bEndOfBlock</i> used as last character
10	bEndOfBlock	1	character terminating TsDataRead
11	fHangup	1	TRUE places the line onhook
12	fCharMode	1	data processing on a per character basis
13	fNoWaitForDialTone	1	Telephone Service waits for dial tone
14	fReadTimeoutReset	1	counter is reset when character is received
15	fWriteTimeoutReset	1	counter reset when character transmitted

openMode

is the data open mode. The values are

Value	Description
0	Convert. Convert an existing voice call to a data call.
1	Accept. Wait for an incoming call to occur.
2	Dial. Place an outgoing data.

fOriginate

is a flag. If true the modem will be the originator. Otherwise it is the answering party.

cTimeout

is the maximum time that is allowed to complete the open after the initial connection before returning status code 11206 ("Timeout").

baudrate

is either 300 or 1200

parity

is the parity control. The values are

Value	Description
0	none
1	even (bit 7 is set or cleared so that there are always an even number of bits set in each byte)
2	odd (bit 7 is set or cleared so that there are always an odd number of bits set in each byte)
3	1 (bit 7 is always set); also known as 'mark'
4	0 (bit 7 is always cleared); also known as 'space'

lineControl

is one of the following values:

Value	Description
0	no flow control
1	XON/XOFF flow control

fLL

is a flag that should be set TRUE by the program if it is running on a single partition version of the operating system.

fEndOfBlock

is a flag. TRUE means the byte value in *bEndOfBlock* will be used as the last character in any *TsDataRead* operation.

bEndOfBlock

is the character that will terminate a *TsDataRead* operation when *fEndOfBlock* is TRUE.

fHangup

is a flag. TRUE means the line will be placed onhook at the completion of data transmission. Otherwise it will be placed on hold when the data call is closed or terminated.

fCharMode

is a flag. TRUE means data will be processed on a per character basis, rather than on a block basis.

fNoWaitForDialTone

is a flag. TRUE means the Telephone Service will not wait for a dial tone before dialing.

fReadTimeoutReset

is a flag. TRUE means the Telephone Service will reset its timeout counter for the *TsDataRead* operation whenever a character is received.

fWriteTimeoutReset

is a flag. TRUE means the Telephone Service will reset its timeout counter for the *TsDataWrite* operation whenever a character is transmitted.

This page intentionally left blank

Description

The *Decoding Offsets Table* is used by the keyboard process to identify the offset, from the beginning of the Translation Data Block, of each entry in the Decoding Table. (See Decoding Table.)

In the Decoding Offsets Table, each entry contains the data block offset of a corresponding entry in the Decoding Table. For example, the fifth entry in the Decoding Offsets Table contains the offset of the fifth entry in the Decoding Table.

A program can use the Decoding Offsets Table to determine the size of a Decoding Table entry. To do so, the program subtracts the value in the corresponding entry of the Decoding Offsets Table from the next entry in the same table. As an example, assume that entries 5 and 6 in the Decoding Offsets Table contains the values 30 and 45, respectively. In this case, the program can determine that entry 5 in the Decoding Table is 15 bytes in length. If the difference between two adjacent entries is 0, no corresponding entry exists in the Decoding Table.

The Decoding Table has 256 entries; therefore, the Decoding Offsets Table has 257 entries. The 257th entry points to the first byte after the Decoding Table so that the size of the 256th entry can be computed.

The Decoding Offsets Table can reside in a translation data block. The data block offset of the Decoding Offsets Table is located in the *oDecodingOffsets* field of the Translation Data Block Header. The *cKeys* field of the same header contains the number of entries (minus 1) in the Decoding Offsets Table.

The Decoding Offsets Table only exists if a Decoding Table is present. If the Decoding Offset Table does not exist, the *oDecodingOffsets* field of the Translation Data Block Header is 0.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Decoding Offsets Table

Offset	Field	Size (bytes)	Description
0	oEntry1	2	offset of first entry in Decoding Table
2	oEntry2	2	offset of second entry in Decoding Table
.			
.			

The *Decoding Offsets Table* contains 257 entries.

oEntry1

is the offset of the first entry in the Decoding Table.

oEntry2

is the offset of the second entry in the Decoding Table.

.

.

.

and so forth.

This page intentionally left blank.

Description

The *Decoding Table* identifies the unencoded values that are responsible for producing a character code. The first table entry identifies the unencoded values that yield a character code of 0; the second table entry identifies the unencoded values that yield a character code of 1, and so forth.

The Decoding Table can contain up to 256 entries. Each entry contains both the upstroke unencoded values and the downstroke unencoded values that generate a character code.

The Decoding Table can reside in a translation data block. This data block, in turn, resides in the file NlsKbd.sys and in memory. The Decoding Offsets Table contains the data block offsets of entries in the Decoding Table. (See the Decoding Offsets Table.) The *cKeys* field of the Translation Data Block Header contains the number of entries in the Decoding Table.

If the Decoding Table is not present in the Translation Data Block, the *oDecodingOffsets* field of the Translation Data Block Header is 0.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Decoding Table

Offset	Field*	Size (bytes)	Description
0	unencValues0	varies	values that produce character code 0
varies	unencValues2	varies	values that produce character code 1
.			
.			
.			

*Decoding Tables contain a variable number of entries.

unencValues1

is an entry containing the unencoded values that produce a character code of 0.

unencValues2

is an entry containing the unencoded values that produce a character code of 1.

.

.

.

and so forth.

This page intentionally left blank.

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

The file management system uses a *Device Control Block (DCB)* to name, locate, and manage each direct-access device under its control. Information contained in the DCB is obtained in the following ways:

- generation at system build
- derivation from the Volume Home Block
- dynamic calculation by the operating system

For a disk, the information may include the number of disk tracks, the number of sectors per track, and so forth. The DCB points to a chain of I/O blocks.

The DCB shown next is supported by CTOS/XE, Version 3.0 and higher operating systems. For a description of the DCB supported by earlier operating system versions, see "Device Control Block (pre-3.0)," later in this chapter.

Related File Structures

- Device Control Block (pre-3.0)
- Directory Entry in a Master File Directory Block
- File Entry in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

Offset	Field	Size (bytes)	Description
0	fMountable	1	device can have a valid CTOS file system
1	fNonSharable	1	device is/is not sharable
2	fRemovable	1	device is a removable medium
3	fOnline	1	TRUE if drive has just become ready
4	fAttention	1	device has become ready or not ready
5	bCacheControl	1	caching is enabled or disabled
6	sbName	13	device name
19	devPassword	13	device password
32	controllerNum	1	device controller number
33	unitNum	1	device physical unit number
34	state	1	zero means idle; non-zero, device specific
35	unitStatus	1	ready, inoperable, or unavailable
36	deviceClass	1	hardware controller type
37	userCount	1	number of device or file system opens
38	oVhb	2	nonzero means a CTOS volume is mounted
40	oQueuedlob	2	nonzero means I/O is pending
42	oActivelob	2	nonzero means I/O is in-progress
44	lfaMax	4	device logical address space size (bytes)
48	chain	2	is used by file system
50	verifyKey	2	checks type of resource open
52	romClass	1	is used by file system
53	romOrdinal	1	is used by file system
54	reserved	2	
56	retryLimit	2	retries before error considered a hard error
58	softErrorCount	2	recovered errors
60	hardErrorCount	2	unrecoverable errors
62	reserved	34	

fMountable

is a flag set to TRUE to indicate that the device may have a CTOS file system.

fNonSharable

is a flag set to FALSE if the device may be used concurrently by multiple users and TRUE if only one user at a time may use the device (for example, 9-track or QIC tape).

fRemovable

is a flag set to TRUE if the device has removable medium.

fOnline

is an internal flag set to TRUE when the device has become ready and may need to have its operating characteristics programmed before any data transfer operations.

fAttention

is a flag used by the file system to determine if the device is ready or not ready. *fAttention* is set in an interrupt handler and is cleared by the MassIO process upon mount or dismount.

bCacheControl

indicates whether caching is enabled or disabled for this device.

sbName

is the name of the device (for example, D0). The first byte is the character string length.

sbPassword

is the device password. The first byte is the character string length.

controllerNum

is the controller number of the device.

unitNum

is the physical unit number of the device.

state

is the state of the device. Zero indicates the device is idle; any nonzero value is device specific.

unitStatus

is the gross operational status of the device. The status can be one of the following values:

Value	State
0	ready
1	not ready (inoperable)
2	locked (unavailable)

deviceClass

is used by the file system to differentiate different classes of hardware controllers. Currently assigned values are:

Value	Device class
0	ST-506 disk devices controlled by a WD-1010/2010
1	floppy diskettes controlled by a WD-2797
2	Storage Module Disk (SMD) controller on a shared resource processor
3	reserved
4	SCSI devices controlled by an NCR 53C90
5	floppy diskettes controlled by a NEC 765
6	SCSI devices controlled by a Unisys ISK Gate Array
8	Memory Disk
10	Disk cache
12	SCSI devices controlled by an NCR 53C94

userCount

is the count of users that have opened the device or files contained in the file system that resides on the device.

oVhb

is the offset of the Volume Home Block (VHB) for the device if a file system is mounted. A zero value indicates that no file system is mounted.

oQueuedIob

is the offset of a linked list of one or more I/O Blocks (IOB's) that are awaiting service at the device. A zero value indicates that no I/O requests are awaiting initiation.

oActiveIob

is the offset of a linked list of one or more I/O Blocks (IOB's) for which operations have been commenced at the device. A zero value indicates that no I/O requests are in progress.

lfaMax

is the size in bytes of the logical address space of the device.

chain

is an internal field used by the file system.

verifyKey

is used by the file system to create file handles for checking the type of resource that is open.

romClass

is an internal field used by the file system.

romOrdinal

is an internal field used by the file system.

retryLimit

is the number of times a retry is attempted after a device error before it is deemed an uncorrectable (hard) error. Some types of errors on some devices use an internal retry algorithm that may retry more or less times than this field indicates.

softErrorCount

is the count of recovered I/O errors.

hardErrorCount

is the count of unrecovered I/O errors.

This page intentionally left blank

Caution: *This system structure is for restricted use only and has been changed for future releases of the operating system. This version of the Device Control Block is used by workstation operating systems only.*

Description

The file management system uses a *Device Control Block (DCB)* to name, locate, and manage each direct-access device under its control. Information contained in the DCB is obtained in the following ways:

- generation at system build
- derivation from the Volume Home Block
- dynamic calculation by the operating system

For a disk, the information may include how many tracks, the number of sectors per track, and so forth. The DCB points to a chain of I/O blocks.

The DCB shown next is supported by operating systems prior to CTOS/XE 3.0. (See the description of *CurrentOsVersion* in Chapter 3, "Operations." The pre-3.0 operating systems have internal version numbers less than 12.0.) For a description of the DCB supported by CTOS/XE 3.0 and higher operations systems, see "Device Control Block," earlier in this chapter.

Related File Structures

- Device Control Block (pre-3.0)
- Directory Entry in a Master File Directory Block
- File Entry in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

Device Control Block (pre-3.0)

Offset	Field*	Size (bytes)	Description
0	fMountable*	1	TRUE means device has valid file system
1	fNonSharable*	1	device is/is not sharable
2	fDoubleDensity*	1	TRUE means a is a dual density diskette
2	fRemovable*	1	TRUE means a removable medium
3	fNoMultiTrack*	1	multitrack read/writes not supported
4	fAttention*	1	device has come ready or not ready
5	reserved	1	
6	sbName*	13	device name
19	sbPassword*	13	device password
32	controllerNum*	1	device controller number
33	unitNum*	1	device physical unit number
34	state	1	zero means idle; non-zero, device specific
35	unitStatus	1	ready, inoperable, or unavailable
36	deviceClass	1	hardware controller type
37	userCount	1	number of device or file system opens
38	oVhb	2	nonzero means a CTOS volume is mounted
40	olobFirst	2	offset to device I/O Block head
42	olobActive	2	offset to the currently active IOB list
44	lfaMax	4	device logical address space size (bytes)
48	lfaMask	2	mask used to verify sector-aligned lfa
50	verifyKey	2	checks the type of resource open
52	reserved	1	
53	fHdcRecovered	1	TRUE means successful error recovery
54	hdcRetries	2	number of retries before error recovery
56	setRetryCnt	2	retries after soft I/O error before logging
58	softErrorCnt	2	count of recovered I/O errors
60	hardErrorCnt	2	accumulated unrecovered error count
62	currentCylinder	2	is set after every successful seek
64	sectorSizeCode	1	is the sector size
Workstations only (offsets 65 through 67)			
65	gapLength	2	controller parameter
67	dataLength	1	floppy controller parameter
Shared resource processors only (offsets 65 through 67)			
65	gapLength	1	ST-506 disk controller parameter
66	writePrecompCyl	1	write precompensation starting cylinder
67	stepRate	1	interval between seek step pulses
<i>(continued)</i>			

Workstations and shared resource processors (offsets 68 through 74)

68	bytesPerSector	2	bytes per disk sector
70	sectorsPerTrack	2	sectors per track
72	tracksPerCylinder	2	tracks per cylinder
74	cylindersPerDisk	2	cylinders per disk

Workstations only (offsets 76 through 77)

76	fInhibitEcc	1	for SMD drives (IWS only)
77	reserved	3	

Shared resource processors only (offsets 76 through 79)

76	fEccFormat	1	TRUE if drive can use ECC
77	spiralFactor	1	for SMD drives
78	fOnline	1	TRUE if drive is online
79	fInhibitEcc	1	for SMD drives

*device independent fields

fMountable

is a flag set to TRUE to indicate that the device may have a CTOS file system.

fNonSharable

is a flag set to FALSE if the device may be used concurrently by multiple users and TRUE if only one user at a time may use the device (for example, 9-track or QIC tape).

fDoubleDensity (workstations only)

is a flag that is TRUE if the device is a diskette and the medium is dual density.

fRemovable (shared resource processors only)

is a flag set to TRUE if the device's medium is removable.

fNoMultiTrack

is a flag that is TRUE if the controller does not support multiple track read/write operations.

Device Control Block (pre-3.0)

fAttention

is a flag used by the file system to determine if the device is ready or not ready. *fAttention* is set in an interrupt handler and is cleared by the MassIO process upon mount or dismount.

sbName

is the name of the device (for example, D0). The first byte is the character string length.

sbPassword

is the device password. The first byte is the character string length.

controllerNum

is the controller number of the device.

unitNum

is the physical unit number of the device.

state

is the state of the device. The state can be one of the following values:

Value	State
0	idle
1	seeking
2	busy (reading or writing)

unitStatus

is the gross operational status of the device. The status can be one of the following values:

Value	State
0	ready
1	not ready or inoperable

deviceClass

is used by the file system to differentiate different classes of hardware controllers. Currently assigned values are:

Value	Device class
0	Winchester on either an AWS or an IWS that has the nonSMD controller
1	a floppy on any AWS or on an IWS that has the nonSMD controller
2	Winchester on the IWS controller
3	floppy on the IWS controller
4	SMD on the IWS controller

userCount

is the count of users that have opened the device or files contained in the file system that resides on the device.

oVhb

is the offset of the Volume Home Block (VHB) for the device if a file system is mounted. A zero value indicates that no file system is mounted.

Device Control Block (pre-3.0)

oIobFirst

is the offset to the head of the I/O Block (IOB) list for the device.

oIobActive

is the offset of a linked list of one or more I/O Blocks (IOB's) for which operations have been commenced at the device. A zero value indicates that no I/O requests are in progress.

lfaMax

is the size (in bytes) of the logical address space of the device.

lfaMask

is the mask used for verifying that the logical file address (lfa) is sector-aligned.

verifyKey

is used by the file system to create file handles for checking the type of resource that is open.

fHdcRecovered

is a flag that is TRUE if the recovery from a non-SCSI disk error is successful. Recovery is logged in the system Log file.

hdcRetries

is the number of retries before successful recovery from a non-SCSI disk error.

setRetryCnt

is the number of times an operation is retried after a soft I/O disk error before the error is logged.

softErrorCnt

is the count of recovered I/O errors.

hardErrorCnt

is the count of unrecovered I/O errors.

currentCylinder

is set after every successful seek. *currentCylinder* is used by the queueing algorithm.

sectorSizeCode

is the sector size. The size can be one of the following values:

Value	Sector size
0	use bytes per sector field (must be less than 256)
1	256 bytes
2	512 bytes
3	1024 bytes

Workstations only (offsets 65 through 67)

gapLength

is a parameter value to the controller hardware that indicates the length of the unused area between sectors of a formatted disk.

dataLength

is a floppy controller parameter that indicates the length of the used area in a sector.

Device Control Block (pre-3.0)

Shared resource processors only (offsets 65 through 67)

gapLength

is a parameter value to the controller hardware that indicates the length of the unused area between sectors of a formatted disk.

writePrecompCyl

is the cylinder at which write precompensation starts.

stepRate

is an encoded number that represents the time interval between seek step pulses for disks that are controlled by a WD-1010 or WD-2010.

Workstations and shared resource processors (offsets 68 through 74)

bytesPerSector

is a disk size parameter indicating the number of bytes per sector..

sectorsPerTrack

is a disk size parameter indicating the number of sectors per track.

tracksPerCylinder

is a disk size parameter indicating the number of tracks per cylinder.

cylindersPerDisk

is a disk size parameter indicating the number of cylinders per disk.

Workstations only (offsets 76 through 77)

fInhibitEcc

is a flag that is TRUE if the device (typically SMD) cannot be configured to use error correction code (ECC).

Shared resource processors only (offsets 76 through 79)

fEccFormat

is a flag that is TRUE if the device can be formatted to use ECC.

spiralFactor

is the sector number offset between adjacent tracks for disks (typically SMD) that use a spiral layout of data sectors to improve performance.

fOnline

is a flag that is TRUE if the drive is online.

fInhibitEcc

is a flag that is TRUE if the device (typically SMD) cannot be configured to use ECC.

Description

The *Diacritic Masks Table* identifies the emulation or translation tables in which a key is defined as the first member of a diacritic pair.

Each entry in the table corresponds to a single key. In an emulation data block and a translation data block, the first entry corresponds to the key with a keyboard code of 0; the second entry corresponds to the key with a keyboard code of 1, and so forth.

Within an entry, the position of the set bit identifies the emulation or translation table in which the key is the start of a diacritic pair. For example, if the first bit (bit 0) in entry 40 is set, a keyboard code of 40 represents the start of a diacritic pair in the first emulation or translation table.

The Diacritic Masks Table is a support table that resides in a translation or emulation data block. The *cKeys* field of the Translation or Emulation Data Block Header contains the number of entries in the Diacritic Masks Table. The *oDiacritMasks* field of either header contains the offset, from the beginning of the data block, of the Diacritic Masks Table.

Note that the Diacritic Masks Table does not exist unless diacritics are defined. If no Diacritic Masks Table is present, *oDiacritMasks* is 0.

To create or modify a diacritic masks table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Diacritic Masks Table

Offset	Field*	Size (bytes)	Description
0	diacritMask0	2	mask value for key 0
2	diacritMask1	2	mask value for key 1
.			
.			
.			

*A diacritic masks table contains a variable number of entries.

diacritMask0

is a mask value that identifies the emulation or translation table in which a keyboard code of 0 represents a diacritic key.

diacritMask1

is a mask value that identifies the emulation or translation table in which a keyboard code of 1 represents a diacritic key.

.

.

.

This page intentionally left blank.

Description

In a translation data block, the *Diacritics Table* identifies the value produced when a diacritic key pair is pressed. This value can be either a character code or the index of a string in the Multibyte Strings Table. In an emulation data block, the *Diacritics Table* identifies either the chord state and emulated keyboard code produced when a diacritic key pair is pressed or the index of a string in the Multibyte Strings Table.

The Diacritics Table is not present unless diacritic keys have been defined. For details on diacritic keys, see "Keyboard and I-Bus Management" in the *CTOS Operating System Concepts Manual*.

The Diacritics Table can reside in both a translation data block and an emulation data block. The *oDiacritics* field of the Translation or Emulation Data Block Header contains the offset, from the beginning of the data block, of the Diacritics Table. The *cDiacritics* field of either header contains the number of entries in the Diacritics Table. If no Diacritics Table is present, *oDiacritics* is 0.

To create or modify a diacritics table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Diacritics Table

Offset	Field*	Size (bytes)	Description
0	diacriticDesc1	9	entry for first diacritic pair
9	diacriticDesc2	9	entry for second diacritic pair
.			
.			
.			

* A Diacritics Table contains a variable number of entries.

diacriticDesc1

is the entry for the first diacritic pair. All entries in the Diacritics Table have the following format:

Offset	Field	Size (bytes)	Description
0	firstTableNumber	1	The number of the table that is selected when the first key is pressed.
1	firstKeyValue	1	The raw value of the first key in the diacritic pair.
2	secondTableNumber	1	The number of the table that is selected when the second key is pressed.
3	secondKeyValue	1	The unencoded value of the second key in the diacritic pair.

Diacritics Table

(continued)

Offset	Field	Size (bytes)	Description
4	resultingValue	4	In an emulation data block, the low-order word is the emulated chord state, and the high-order word is the emulated keyboard code. In a translation data block, this field contains the character code. In either case this also may be an index into the Multibyte Strings Table.
8	stringFlag	1	A flag that is TRUE if the value in <i>resultingValue</i> is the index of a string in the Multibyte Strings Table.

diacriticDesc2

is the entry for the second diacritic pair.

.
.
.

and so forth.

Directory Entry in Master File Directory Block

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

When a disk volume is initialized, the volume control structures for that volume are created. The volume control structures include the following:

- Volume Home Block
- File Header Block
- Master File Directory
 - directories
- Allocation Bit Map

The volume control structures allow the file management system to manage the space on a volume not dedicated to the control structures themselves. (For details on all the volume control structures and their relationships, see "File Management" in the *CTOS Operating System Concepts Manual*.)

For each directory on a volume, there is an entry in the Master File Directory. Each block of the Master File Directory contains up to 14 directory entries. The structure that follows shows the format of one such directory entry.

Directory Entry in Master File Directory Block *(continued)*

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- File Entry in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

(continued) **Directory Entry in Master File Directory Block**

Offset	Field	Size (bytes)	Description
0	sbDirEntryName	13	directory name
13	sbPassword	13	directory password
26	lfaBase	4	points to first block for this directory
30	cPages	2	count of blocks for this directory
32	defaultAccessCode	1	default protection level for files created
33	lruCnt	2	in-memory directory access count

sbDirEntryName

is the directory name. The first byte is the character string size.

pbPassword

is the directory password. The first byte is the character string size.

lfaBase

is the address of the first block of this directory.

cPages

is the count of blocks for this directory.

defaultAccessCode

is the default protection level for files created in this directory.

lruCnt

is the in-memory directory access count (Version 9.X operating systems only).

This page intentionally left blank

Description

The *Emulation Data Block Header* is located at the beginning of an emulation data block. Information in the Emulation Data Block Header includes the memory offsets of various supporting tables within the data block, the number of entries within these tables, the ID of the attached keyboard, and the ID of the target keyboard.

Examples of emulation data block tables include the Allowed States Table and the Special Keys Table. Each supporting table within an emulation data block is also described in this chapter.

The Emulation Data Block Header provides the keyboard process with the information it needs to access the tables within the emulation data block.

For a given partition, the address of the Emulation Data Block Header can be found in the Partition Keyboard Information Structure.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Emulation Data Block Header

Offset	Field	Size (bytes)	Description
0	emulDataBlock-Signature	2	signature of the Emulation Data Block ("HE")
2	emulDataBlockID	2	ID of the Emulation Data Block
4	cbEmulHeader	2	size of the Emulation Data Block Header
6	emulDataBlock-Version	2	version number of emulation data block
8	targetKeyboard	2	ID of target keyboard
10	cChords	2	count of entries in Chords Table
12	oChords	2	offset of Chords Table
14	cSpecialKeys	2	count of entries in Special Keys Table
16	oSpecialKeys	2	offset of Special Keys Table
18	cConditions	2	count of entries in Conditions Table
20	oConditions	2	offset of Conditions Table
22	cKeys	2	count of emulation keys
24	oWorkArea	2	offset of work area for OS
26	oAllowedStates	2	offset of Allowed States Table
28	oDiacritMasks	2	offset of Diacritic Masks Table
30	cDiacritics	2	count of diacritics in Diacritics Table
32	oDiacritics	2	offset of Diacritics Table
34	oStringMasks	2	offset of Multibyte String Masks Table
36	cStringOffsets	2	count of entries in Multibyte String Offsets Table
38	oStringOffsets	2	offset of Multibyte String Offsets Table
40	cEmulTablesDescs	2	number of emulation tables described
42	pRgEmulTableDescs	2	offset of emulation table descriptor array
44	cLedData	2	count of entries in Emulation LEDs Table
46	oLedData	2	offset of Emulation LEDs Table
48	cbChecked	2	number of bytes included in checksum
50	checksum	4	checksum of data in data block

emulDataBlockSignature

is "EH", the signature of the Emulation Data Block.

emulDataBlockID

is the ID of the Emulation Data Block. The high-order byte is always 71h, and the low-order byte is the ID of the corresponding keyboard.

cbEmulHeader

is the size of the Emulation Data Block Header.

emulDataBlockVersion

is the version number of the Emulation Data Block.

targetKeyboard

is the ID of the target keyboard.

cChords

is the count of entries in the Chords Table.

oChords

is the offset, from the beginning of the Emulation Data Block Header, of the Chords Table. (See the Chords Table.)

cSpecialKeys

is the count of entries in the Special Keys Table.

oSpecialKeys

is the offset, from the beginning of the Emulation Data Block Header, of the Special Keys Table. (See the Special Keys Table.)

cConditions

is the count of condition entries in the Conditions Table.

oConditions

is the offset, from the beginning of the Emulation Data Block Header, of the Conditions Table. (See the Conditions Table.)

cKeys

is the count of entries in the Emulation Table(s), the Allowed States Table, the Diacritic Masks Table, the Multibyte String Masks Table, and the Command Masks Table.

oWorkArea

is the offset of a work area that is reserved for use by the operating system.

oAllowedStates

is the offset, from the beginning of the Emulation Data Block Header, of the Allowed States Table. (See the Allowed States Table.)

oDiacritMasks

is the offset, from the beginning of the Emulation Data Block Header, of the Diacritic Masks Table. (See the Diacritic Masks Table.)

cDiacritics

is the count of diacritic key entries in the Diacritics Table.

oDiacritics

is the offset, from the beginning of the Emulation Data Block Header, of the Diacritics Table. (See the Diacritics Table.)

oStringMasks

is the offset, from the beginning of the Emulation Data Block Header, of the Multibyte String Masks Table. (See the Multibyte String Masks Table.)

cStringOffsets

is the count of entries in the Multibyte String Offsets Table.

oStringOffsets

is the offset, from the beginning of the Emulation Data Block Header, of the Multibyte String Offsets Table. (See the Multibyte String Offsets Table.)

cEmulTablesDescs

is the number of emulation tables described in the Descriptor Array. This array contains the data block offset of each emulation table in the Emulation Data Block.

pRgEmulTableDescs

is the offset, from the beginning of the Emulation Data Block Header, of the emulation table descriptor array.

cLedData

is the count of LEDs on the emulated keyboard.

oLedData

is the offset, from the beginning of the Emulation Data Block Header, of the Emulation LEDs Table. (See the Emulation LEDs Table.)

(continued)

Emulation Data Block Header

cbChecked

is the number of bytes, from the beginning of the Emulation Data Block Header, that were included in the checksum.

checksum

is an internal value used to verify the integrity of the Emulation Data Block.

This page intentionally left blank.

Description

The *Emulation LEDs Table* allows the end user to remap LED identifiers. If an LED key is not present on a particular keyboard, the Emulation LEDs Table can be used to map the LED key to one that does exist. Thus, when the application specifies an LED identifier in a call to `SetKbdLed`, the LED specific to the attached keyboard is activated. In this manner, the Emulation LEDs Table provides hardware-independence for existing applications.

The Emulation LEDs Table contains 16 one-byte entries, each of which corresponds to a single LED key. The order of the entries cannot be changed. For example, the first entry corresponds to the **F10** key, the second entry corresponds to the **F9** key, and so forth. In addition, the entries for the LED keys correspond to the LEDs as listed in the call `SetKbdLed`. For example, offset 0 corresponds to `iLed 0` for the **F10** key.

If an LED key is remapped, its entry contains 1 plus the `iLed` value of the LED key that will actually be referenced. For example, suppose the entry for the **num Lock** key contains a value of 4. When an application specifies the `iLed` value for the **num Lock** key in a call to `SetKbdLed`, the LED key represented by `iLed 3` (i.e., 4 minus 1) is affected.

If an entry contains 0, the LED key is not remapped. As an example, suppose the entry for the **F9** key contains a value of 0. When the application specifies the `iLed` value for the **F9** key in a call to `SetKbdLed`, the **F9** key is still turned on or off.

When an LED key with an entry value of `FFh` is specified in a call to `SetKbdLed`, no LED key is turned on or off.

The Emulation LEDs Table resides in an emulation data block. The *oLedData* field of the Emulation Data Block Header contains the data block offset of the Emulation LEDs Table. Similarly, the *cLedData* field of the same header contains the number of entries in the Emulation LEDs Table.

To create or modify an emulation LEDs table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Emulation LEDs Table

Offset	Field*	Size (bytes)	Description
0	F10entry	1	entry for the F10 key.
1	F9entry	1	entry for the F9 key.
2	F8entry	1	entry for the F8 key.
3	F3entry	1	entry for the F3 key.
4	F2entry	1	entry for the F2 key.
5	F1entry	1	entry for the F1 key.
6	lockEntry	1	entry for the Lock key.
7	overtimeEntry	1	entry for the Overtime key.
8	comm1Entry	1	entry for the first comm LED key.
9	comm2Entry	1	entry for the second comm LED key.
10	comm3Entry	1	entry for the third comm LED key.
11	comm4Entry	1	entry for the forth comm LED key.
12	comm5Entry	1	entry for the fifth comm LED key.
13	numLockEntry	1	entry for the Num Lock key.
14	CtosLockEntry	1	entry for the CTOS Lock key.

F10entry

is the entry for the **F10** key.

F9entry

is the entry for the **F9** key.

F8entry

is the entry for the **F8** key.

F3entry

is the entry for the **F3** key.

F2entry

is the entry for the **F2** key.

F1entry

is the entry for the **F1** key.

lockEntry

is the entry for the **Lock** key.

overtimeEntry

is the entry for the **Overtime** key.

comm1Entry

is the entry for the first communication LED key.

comm2Entry

is the entry for the second communication LED key.

comm3Entry

is the entry for the third communication LED key.

comm4Entry

is the entry for the forth communication LED key.

comm5Entry

is the entry for the fifth communication LED key.

numLockEntry

is the entry for the **Num Lock** key on the SG-102-K.

CtosLockentry

is the entry for the **CTOS Lock** key on the SG-102-K.

Description

An *emulation table* maps the chord state and unencoded values on the attached keyboard to the chord state and unencoded values necessary to produce the same character on a different keyboard.

As an example, assume that the attached keyboard is emulating a K1 keyboard. If a keystroke produces the keyboard code 7Eh, the entry at index 7E in an emulation table specifies the equivalent chord state/key combination on the K1 keyboard.

A maximum of 16 emulation tables may be present in an emulation data block. The data block offset of an emulation table is located in the Table Descriptor Array.

For more information on emulation tables, see "Keyboard and I-Bus Management" in the *CTOS Operating System Concepts Manual*.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Emulation Table

Offset	Field*	Size (bytes)	Description
0	tableEntry1	3	1st entry of emulation table
3	tableEntry2	3	2nd entry of emulation table
.			
765	tableEntry256	3	256th entry of emulation table

*An emulation table presently contains 256 entries.

tableEntry1

is the 1st entry in an emulation table. Each entry has the following format:

Offset	Field	Size (bytes)	Description
0	chordIdentifier	2	A set of mask values that identify the emulated chord state. If <i>chordIdentifier</i> is FFFFh, the chord state is not changed during emulation.
2	keyboardCode	1	The emulated keyboard code. If <i>keyboardCode</i> is FFh, the key is not supported.

tableEntry2

is the 2nd entry in an emulation table.

.

.

.

tableEntry256

is the 256th entry in an emulation table.

Description

The *Expanded Date/Time format* represents the system date and time using discreet fields for the year, month, day of month, day of week, hour, minute, and second.

The CompactDateTime and ExpandDateTime operations can be used to convert between the Expanded Date/Time format and the System Date/Time format. Using the NlsParseTime or ParseTime operation, the caller can convert a string containing the date and time to the Expanded Date/Time format. (For details on all the formats in which the system date and time can be represented, see "Utility Operations" in the *CTOS Operating System Concepts Manual*.)

Related Timer and Interrupt Structures

- Low Memory Allocation
- System Date/Time Structure
- Timer Pseudointerrupt Block
- Timer Request Block Format

This page intentionally left blank

(continued)

Expanded Date/Time Format

Offset	Field	Size (bytes)	Description
0	year	2	1952-2042 (0=null date/time)
2	month	1	0-Jan, 11-Dec
3	dayOfMonth	1	1-31
4	dayOfWeek	1	0-Sun, 6-Sat
5	hour	1	0-23
6	minute	1	0-59
7	second	1	0-59

year

is the year (1952 to 2042). 0 = null date/time.

month

is the month. The values of the months are 0 through 11, for January through December.

dayOfMonth

is the day of the month. The day values are 1 through 31.

dayOfWeek

is the day of the week. The values are 0 through 6, for Sunday through Saturday.

hour

is the hour of the day. The values are 0 through 23.

minute

is the minute. The values are 0 through 59.

second

is the second. The values are 0 through 59.

This page intentionally left blank

Description

The *Extended Partition Descriptor* is an application partition data structure. It contains environment information on the current run file in the application partition. For example, it contains the address of the Video Control Block, the status and contents of the type-ahead buffer, the status of the submit file, and a description of the virtual character map. A pointer to this structure is returned by GetPStructure (code 0).

Related Partition Structures

- Application System Control Block
- Batch Control Block
- Partition Configuration Block
- Partition Descriptor
- Variable Length Parameter Block

This page intentionally left blank

(continued)

Extended Partition Descriptor

Offset	Field	Size (bytes)	Description
0	sbCurRunFileSpec(79)	79	current run file specification
79	sbExitRunFileSpec(79)	79	current exit run file specification
158	sbExitRunFilePswd(13)	13	exit run file password
171	exitRunFilePriority	1	exit run file priority
172	prgbBufOut	4	address of type-ahead buffer
176	ibBufOutPut	2	next keystroke to be placed in buffer
178	ibBufOutTake	2	next keystroke to be read from buffer
180	fBufOutOverflow	1	indicates that type-ahead buffer is full
181	pRqKbdBlocked	4	address of outstanding keyboard request
185	fReadOrPeekReceived	1	<i>pRqKbdBlocked</i> is set
186	fResetDisabled	1	Action-Finish disabled for the user number
187	fChAction	1	next byte contains last action code typed
188	chAction	1	last action code typed
189	fUnencoded	1	in unencoded mode
190	ledState	1	bit encoded state of the keyboard LEDs
191	prgbSubBuf	4	address of 512-byte submit file work area
195	ibNext	2	index to next byte in submit file work area
197	ibMac	2	total bytes currently in the work area
199	lfaSubEOF	4	size in bytes of submit file
203	reserved	2	
205	userNumFhSub	2	submit file handle's user number
207	lfaSubFile	4	number of bytes read from submit file
211	fhSub	2	submit file handle
213	bContinue	1	submit file continuation character
214	fActionMode	1	escape to real keyboard from submit file
215	stateSysIn	1	state of keyboard filter process
216	recordState	1	state of submit file
217	pRqSave	4	address of the saved user request
221	pCharMap	4	address of virtual character map
225	sCharMap	2	size of virtual character map
227	pVcb	4	address of Video Control Block
231	pRqKbd	4	address of keyboard filter's request block
235	prgpVidLine	4	Lock(Unlock)Video character map address
239	prgbEventTypeBufOut	4	address of I-Bus input event buffer
243	pPntDeviceData	4	address of pointing device input data

sbCurRunFileSpec

is the full file specification of the current run file. The first byte is the specification string length.

sbExitRunFileSpec

is the full file specification of the exit run file. The first byte is the specification string length.

sbExitRunFilePswd

is the exit run file password. The first byte is the password string length.

exitRunFilePriority

is the priority of the exit run file.

prgbBufOut

is the address of the type-ahead circular buffer (I-Bus data array). (See the description of *prgbEventTypeBufOut* at offset 239 in this structure.)

ibBufOutPut

is the index to the next keystroke to be placed in the buffer.

ibBufOutTake

is the index to the next keystroke to be read from the buffer.

fBufOutOverflow

is a flag that is set to TRUE if the buffer is full.

pRqKbdBlocked

is the address of the current outstanding keyboard request.

fReadOrPeekReceived

is a flag that is TRUE if *pRqKbdBlocked* is set.

fResetDisabled

is a flag that is TRUE if **Action-Finish** is disabled for this user number.

fChAction

is a flag that is TRUE if the next byte contains the last action code typed. *fChAction* is set when an action code (other than **Action-A**, **Action-B**, **Action-Finish**) is typed. The `ReadActionCode` operation clears this flag after reading an action code.

chAction

is the last action code typed.

fUnencoded

is a flag that is TRUE if in unencoded mode.

ledState

is the bit encoded state of the keyboard LEDs.

prgbSubBuf

is the address of the 512-byte submit file work area.

ibNext

is the index to the next byte in the submit file work area.

ibMac

is the total number of bytes currently in the work area.

lfaSubEOF

is the size in bytes of the submit file.

userNumFhSub

is the user number associated with the submit file handle.

lfaSubFile

is the current number of bytes read from the submit file.

fhSub

is the submit file handle.

bContinue

is the submit file continuation character.

fActionMode

is a flag that is TRUE if the submit file input is currently being obtained from the keyboard process rather than from the system input process.

stateSysIn

is the state of the keyboard filter process.

recordState

is the state of the current submit file (record, playback, normal, or waiting for continuation key)

pRqSave

is the address of the saved user request.

pCharMap

is the address of the virtual character map.

sCharMap

is the size in bytes of the virtual character map.

pVcb

is the address of the Video Control Block.

pRqKbd

is the address of the request block for the keyboard filter process.

prgpVidLine

is the address of the character map used by callers of LockVideo and UnlockVideo.

prgbEventTypeBufOut

is the address of the I-Bus input event type buffer. Each word element in this array corresponds to a word element in the I-Bus data array *rgbBufOut* previously described. Each element in *rgbEventTypeBufOut* identifies the originator (such as keyboard or mouse) of the data placed in the corresponding I-Bus data array element.

pPntDeviceData

is the address of the pointing device input data.

This page intentionally left blank

Description

The *Extended Process Control Block (EPCB)* is an extension of the Process Control Block structure described later in this chapter. The EPCB contains fields for specific applications, such as the X-Bus and MS-DOS. An offset to this structure is contained in the PCB.

Related System Structures

- Port Structure
- Process Control Block
- System Configuration Block

This page intentionally left blank

(continued)

Extended Process Control Block

Offset	Field	Size (bytes)	Description
0	<i>oSoftVecHead</i>	2	low memory addresses for this process
2	<i>deltaPriority</i>	1	value added to process priority by Kernel
3	<i>savedPriority</i>	1	process priority (0 to 255)
4	<i>extendedAddressRegister</i>	2	4 bytes for accessing X-Bus memory
6	<i>oVector</i>	2	low memory changes for MS-DOS vectors
8	<i>nwVectors</i>	2	low memory changes for MS-DOS vectors
10	<i>pVectorArea</i>	4	low memory changes for MS-DOS vectors
14	<i>oNextPcb</i>	2	links PCBs by user number

oSoftVecHead

is an offset to the low memory addresses for this process, for example, the System Common Address Table (SCAT) pointers and trap vectors.

deltaPriority

is a value added to the priority of a process by the Kernel before determining which process to dispatch. This value is the same for all processes with a given user number. (See the description of *SetDeltaPriority* in "Operations.")

savedPriority

is the process priority (0 to 255, with 0 the highest).

extendedAddressRegister

are the extra 4 bytes (provided by the Extended Address Register) needed by this real mode (only) process to access the memory of an X-Bus module. (For details, see "X-Bus Management" in the *CTOS Operating System Concepts Manual*.)

oVector

describes how low memory should change when a process switch is made to this process. On a process switch to this process, the value of *oVector* and the values of *nwVectors* and *pVectorArea* (next two fields) change the entire Interrupt Vector Table.

nwVectors

See the description of *oVector*.

pVectorArea

See the description of *oVector*.

oNextPcb

links PCBs by user number to optimize termination.

Description

When a disk volume is initialized, the volume control structures for that volume are created. The volume control structures include the following:

- Volume Home Block
- File Header Block
- Master File Directory
- directories
- Allocation Bit Map

The volume control structures allow the file management system to manage the space on a volume not dedicated to the control structures themselves. (For details on all the volume control structures and their relationships, see "File Management" in the *CTOS Operating System Concepts Manual*.)

Each directory on a volume consists of one or more directory blocks. Hashing techniques determine the directory block into which a file is entered. The structure that follows shows the format of a file entry in a directory block.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Directory Entry in a Master File Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

(continued)

File Entry in a Directory Block

Offset	Field	Size (bytes)	Description
0	cbFileName	1	
1	fileName	cbFileName	
1+cbFileName	fhbNum	2	

cbFileName

is the count of bytes in the file name.

fileName

is the file name. Note that case is preserved in the stored file names, but hashing is case-insensitive.

fhbNum

is the File Header Block (FHB) number. *fhbNum* a relative block index into the file <Sys>Fileheaders.sys, which locates the FHB.

This page intentionally left blank

Description

When a disk volume is initialized, the volume control structures for that volume are created. The volume control structures include the following:

- Volume Home Block
- File Header Block
- Master File Directory
- directories
- Allocation Bit Map

The volume control structures allow the file management system to manage the space on a volume not dedicated to the control structures themselves. (For details on all the volume control structures and their relationships, see "File Management" in the *CTOS Operating System Concepts Manual*.)

There is a *File Header Block (FHB)* for each file. The FHB contains information about the file such as its name, password, protection level, the date/time it was created, the date/time it was last modified, and the disk address and size of each of its disk extents. The FHB is one block in size and can accommodate up to 32 disk extents. If additional extents are needed, another FHB is created.

The SetFileStatus operation sets certain fields in the FHB. (For details, see the description of SetFileStatus in Chapter 3, "Operations.")

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Directory Entry in a Master File Directory Block
- File Entry in a Directory Block
- Master File Directory Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

(continued)

File Header Block

Offset	Field	Size (bytes)	Description
0	checksum	2	ensures validity of FHB contents
2	fileHeaderPageNum	2	block number where this FHB is located
4	sbFileName	51	file name
55	sbPassword	13	file password
68	sbDirName	13	directory in which file was created
81	fileHeaderNum	2	primary/secondary file header number
83	extHeaderNumChain	2	address of next FHB for this file
85	headerSequenceNum	1	sequence number of this FHB
86	fileClass	1	available for use by applications
87	accessProtection	1	file access protection level
88	lfaDirPage	4	directory block containing FHB pointer
92	creationDT	4	date/time file was created
96	modificationDT	4	date/time file was last modified
100	accessDT	4	date/time file was last opened
104	expirationDT	4	available for use by applications
108	fNoSave	1	TRUE means do not back up
109	fNoDirPrint	1	not currently used
110	fNoDelete	1	TRUE for all files IVolume places on disk
111	lfaEndOfFile	4	file length
115	defaultExpansion	4	number of blocks to expand the file
119	freeRunIndex	2	next free extent index in <i>vda</i> and <i>runLength</i>
121	vda(runsPerFhb)	128	addresses of the file disk extents
249	runLength	128	block counts of the disk extent addresses
377	userObjectType	2	file type displayed by an application
379	reserved	51	
430	wlInternational	2	used by file system for internationalization
432	keyboardEnvironment	4	ID of target keyboard
436	modificationSync	4	used to create a unique file identifier

File Header Block

(continued)

Offset	Field	Size (bytes)	Description
440	bCacheControl	1	indicates if caching is enabled/disabled
441	fReadOnly	1	TRUE if this file is opened in read-only mode
442	bDosMagic	1	indicates validity of <i>bDosAttribute</i>
443	bDosAttribute	1	DOS attribute assigned the file, if valid
444	parentFhbNum	2	backlink to directory containing this file
446	reserved	1	
447	objectType	1	a value specifying a nested directory or file
448	rgbApplication(64)	1	application-specific

checksum

ensures that contents of FHB are valid.

fileHeaderPageNum

is the block number of [Sys]<Sys>FileHeaders.sys where this FHB is located. Block numbers for primary and secondary FHBs are numbered consecutively starting with 1. The **Backup Volume** and **Format Disk** utilities compare the value of *fileHeaderPageNum* to the value of *fileHeaderNum* (described below) to determine if the FHB is valid or if it has been corrupted.

sbFileName

is the file name. The first character is the size of the name string.

sbPassword

is the file password. The first character is the size of the password string.

sbDirName

is the name of the directory in which this file was created. The first character is the size of the directory name string.

fileHeaderNum

is the file header number of this file. The numbering scheme of *fileHeaderNum* for primary and secondary file headers is as follows: the first 18 primary file headers are numbered consecutively 1 through 18. Following these numbers, the first 18 secondary file headers are numbered 1 through 18. Then, the next 18 primary file headers are numbered starting with 37, followed by the next 18 secondary file headers starting with 37, and so forth.

Because the corresponding values of *fileHeaderPageNum* (described above) are always consecutive numbers starting with 1, the value of *fileHeaderNum* for a the secondary file header should always be 18 less than the value of *fileHeaderPageNum* for the file. The **Backup Volume** and **Format Disk** utilities compare the value of *fileHeaderPageNum* with the value of *fileHeaderNum* to determine if the FHB is valid or if it has been corrupted.

extHeaderNumChain

is the address of the next FHB for this file. This field is updated only when a file length is changed (using the ChangeFileLength operation) to a new length that would require more than 32 disk extents for the file. Each FHB can accomodate up to 32 disk extents.

headerSequenceNum

is the sequence number of this FHB. If there is more than one FHB, this number indicates the relative position of this FHB to the others in the FHB chain.

fileClass

is a field available for use by applications.

accessProtection

is the protection level of this file. (For details on file protection levels, see "File Management" in the *CTOS Operating System Concepts Manual*.)

lfaDirPage

is the directory block containing the file name string byte count, the file name string, and a pointer to the FHB for this file. Files are hashed to directory blocks based on the value of the file name string. The same hashing technique is used to optimize locating the directory block that contains the specified file name string.

creationDT

is the date and time this file was created.

modificationDT

is the date and time this file was last modified.

accessDT

is the date and time this file was last opened.

expirationDT

is a field available for use by applications.

fNoSave

is a flag that is TRUE if this file is not to be backed up. **Backup Volume** uses this field.

fNoDelete

is a flag that is TRUE if this file is not to be deleted. *fNoDelete*, for example, is TRUE for all files (SysImage.sys, Mfd.sys, and so forth) placed on the disk by **Format Disk**.

lfaEndOfFile

is the file length (address of the last byte written to the file). File byte streams automatically sets this field when a file is closed.

defaultExpansion

is the number of blocks to expand this file (default is 30).

freeRunIndex

is the index of the next free extent in the *vda* and *runLength* arrays described below. A value of 32 means there are no more free extents.

vda(runsPerFhb)

is an array of 32 four-byte virtual disk addresses of the disk extents for this file.

runLength

is an array of 32 four-byte elements, each of which contains a byte count corresponding to a virtual disk address in the *vda* array.

userObjectType

describes the type of this file. This field is set by applications to display the appropriate file type. The values of *userObjectType* are shown below:

Value	Description
0-255	reserved for future use
256-32767	application defined as follows:
256	documents
257	BGP pictures
258	Multiplan spreadsheets
259	phone list

File Header Block

(continued)

Value	Description
260	voice file
261	PDB
262	data image file
263	VDM graphics file
264	vector fonts
265	character fonts
266	raster fonts
267	document text type
268	icon files
269-32766	reserved for future use
32767	zero file type
32768-65536	reserved for customers

wInternational

is a value used by the file system for internationalization.

keyboardEnvironment

is the ID of the target keyboard.

modificationSync

is used by the file system to create a unique file identifier.

bCacheControl

indicates if caching is enabled or disabled.

fReadOnly

is a flag that is TRUE if this file is opened in read mode.

bDosMagic

if this field contains the hexadecimal value SM, the DOS attribute byte (next field) is valid.

bDosAttribute

if valid (see *bDosMagic*), is the attribute (such as read only, archive) assigned to the file by MS-DOS.

parentFhbNum

is the backlink to the directory containing this file.

objectType

is one of the following values:

Value	Description
4	Hierarchical file system (nested) directory
5	File

rgbApplication

is an application-specific field.

This page intentionally left blank.

Description

The *Frame Descriptor* describes a frame displayed to the video device. Fields in the frame descriptor define the following characteristics of a frame:

- where the frame is positioned on the screen
- the frame size and border
- where the next character is to be stored by succeeding applications writing to the frame
- the cursor position if the cursor is visible

Related Video Structures

Video Control Block

This page intentionally left blank

Frame Descriptor

Offset	Field	Size (bytes)	Description
0	iLineStart	1	vertical screen coordinate for frame 0
1	iColStart	1	horizontal screen coordinate for frame 0
2	cLines	1	height of the frame
3	cCols	1	width of the frame
4	iLineLeftOff	1	vertical screen coordinate for appending
5	iColLeftOff	1	horizontal screen coordinate for appending
6	bBorderDesc	1	frame border specification value
7	bBorderChar	1	character used for creating the border
8	bBorderAttr	1	character attribute used for the border
9	iLinePause	1	enables or suppresses screen pause
10	iLineCursor	1	vertical coordinate of cursor
11	iColCursor	1	horizontal coordinate of cursor
12	fDbIHigh	1	TRUE for double high on B24 TWSs
13	fDbIWide	1	TRUE for double wide on B24 TWSs
14	reserved	8	

iLineStart

iColStart

are the vertical and horizontal screen coordinates of the upper left corner of the frame. This is where a character would go if the PutFrameChars operation were called with iLine and iCol = 0.

cLines

cCols

are the height and width of the frame.

iLineLeftOff

iColLeftOff

are used by the Sequential Access Method to record the coordinates at which the next character is to be stored in the frame. The presence of this information in the VCB allows a succeeding application program to append to information displayed by its predecessor.

bBorderDesc

is a byte with bits 0-3 specifying a border just outside the frame on the corresponding side.

Bit	Side
0	Top
1	Right
2	Bottom
3	Left

The border is drawn when the `InitCharMap` operation is executed. The same character with the same character attributes (see the *bBorderChar* and *bBorderAttr* parameters of the `InitVidFrame` operation) is used for all sides and corners.

bBorderChar

is the character to use for borders when the `InitCharMap` operation is executed.

bBorderAttr

is the character attribute to use for borders when the `InitCharMap` operation is executed.

iLinePause

is used by the Sequential Access Method to determine when to prompt the workstation operator to press **Next Page** after a new page of text is scrolled onto the screen. *iLinePause* indicates which line (0-33) is "marked." *iLinePause* is decremented whenever the marked line is scrolled upward. When it is decremented to 0, a message prompting the user is displayed. If *iLinePause* is set to 255 (0FFh) as it is by the `InitVidFrame` operation, the functions described above are suppressed.

iLineCursor

iColCursor

are the vertical and horizontal coordinates within the frame at which the visible cursor is displayed. If *iLineCursor* and *iColCursor* are each set to 255 (0FFh), there is no visible cursor in the frame.

fDblHigh

is a flag. TRUE means the characters are double high on B24 Teller Workstations.

fDblWide

is a flag. TRUE means the characters are double wide on B24 Teller Workstations.

This page intentionally left blank

Description

The *High Sierra Directory Record* provides the CD-ROM Service with the information it needs to access files on a CD-ROM disc. Specifically, it provides a description of the contents in a particular directory or file. Information in the High Sierra Directory Record includes the size of the file or directory, the number of logical blocks that reside in the file or directory, and the date and time that the logical blocks were created.

Additionally, the High Sierra Directory Record allows the CD-ROM Service to control access to a file. For example, the CD-ROM Service can specify whether the file's existence is to be hidden from user queries.

The High Sierra Directory Record is similar in function to the *ISO Directory Record*. However, the High Sierra Directory Record is present on discs that follow the High Sierra format, whereas the ISO Directory Record is present on discs that follow the ISO format.

The CD-ROM operations are described in Chapter 3, "Operations." For the format of the High Sierra Directory Record, see "CD-ROM Service" in the *CTOS Programming Guide*.

Related CD-ROM Structures

ISO Directory Record

This page intentionally left blank

Description

The *ISO Directory Record* provides the CD-ROM Service with the information it needs to access files on a CD-ROM disc. Specifically, it provides a description of the contents in a particular directory or file. Information in the ISO Directory Record includes the size of the file or directory, the number of logical blocks that reside in the file or directory, and the date and time that the logical blocks were created.

Additionally, the ISO Directory Record allows the CD-ROM Service to control access to a file. For example, the CD-ROM Service can specify whether the file's existence is to be hidden from user queries.

The ISO Directory Record is similar in function to the *High Sierra Directory Record*. However, the ISO Directory Record is present on discs that follow the ISO format, whereas the High Sierra Directory is present on discs that follow the High Sierra format.

The CD-ROM operations are described in Chapter 3, "Operations." For the format of the ISO Directory Record, see "CD-ROM Service" in the *CTOS Programming Guide*.

Related CD-ROM Structures

High Sierra Directory Record

This page intentionally left blank

Description

The Log file (Log.sys) is one of the six system files created when you initialize a disk with the **Format Disk** command. (For details on **Format Disk**, see the *CTOS Executive Reference Manual*.) The other files created are the Bad Sector File (BadBlk.sys), the Crash Dump File (CrashDump.sys), the File Header Blocks (FileHeaders.sys), the Master File Directory (Mfd.sys), and the System Image (SysImage.sys). (For an illustration of the relationships of these files to the volume control structures on a disk, see "File Management" in the *CTOS Operating System Concepts Manual*.) Like the other system files, the length of the system Log File is determined when the disk is initialized and can only be changed by reinitializing.

The contents of the Log File are called records. Each record describes an event that is significant in the operation of the cluster. A record can be informational, such as records that inform the viewer of a system boot or of a Tape Service installation. Records can also describe error conditions and provide diagnostic information, such as those that log disk or cluster errors.

The *Log File Record format* shown next is the format the operating system uses to write a variable-length record to the system Log File. The first 31 bytes of each record are header information. Following the header, there is a variable-length trailer, which contains the type-specific information. The record format shows the header (common to all records) as well as the format of the type-specific portion for 13 records generated by various programs.

To write a record to the Log File, your program can use the WriteLog operation, providing a description of the record to be logged. (For details, see the description of WriteLog in Chapter 3, "Operations.") When the record is written, the operating system automatically generates the Log File header portion of the format.

Related File Structures

Device Control Block

Device Control Block (pre-3.0)

Directory Entry in a Master File Directory Sector

File Entry in a Directory Sector

Master File Directory Sector

File Header Block

Standard File Header Format

Standard Record Header Format

Standard Record Trailer Format

User Control Block

Volume Home Block

Volume Home Block (pre-3.0)

(continued)

Log File Record Format

Offset	Field	Size (bytes)	Description
Log File Entry Header			
0	sRecord	1	indicates the size of the record
1	dateTime	4	time the record was created
5	logType	2	describes the record type
7	hardwareType	1	hardware type
8	clusterConfig	1	type of cluster configuration
9	fNoFileSystem	1	indicates if the file system is local or not
10	fCommIOP	1	Comm IOP
11	bPartitionType	1	multipartition or variable partition
12	processorType	1	processor type
13	bMfpId	1	processor Id (shared resource processors)
14	bMySlot	1	slot number (shared resource processors)
15	lineNum	1	line number (see GetClusterStatus)
16	wsId	1	workstation ID (see GetClusterStatus)
17	cSeg	2	number of segments of memory
19	sbUserName	12	name of logged on user
31	typeSpecInfo	varies	information specific to this record (described below for each record type)

Type-Specific Formats

Record Type: System Initialization Error

31	erc	2	status code
33	localErc	2	local status code
35	lineNrOrProcType	2	line number or processor type
37	text	80	text

(continued)

Log File Record Format

(continued)

Offset	Field	Size (bytes)	Description
Record Type: Disk Error (pre-3.0)			
31	erc	2	status code
33	retryCnt	1	retry count
34	fRecovered	1	recovered
35	deviceClass	1	device class
36	unitNum	1	unit number
37	command	9	command
46	status	8	drive status
54	sbVolName	13	volume name
67	scsiStatus	14	SCSI status
Record Type: Tape Error			
31	erc	2	status code
33	retryCnt	1	retry count
34	fRecovered	1	recovered
35	deviceClass	1	device class of tape drive
36	unitNum	1	unit number
37	status	8	status
Record Type: System Crash			
31	crashCodes	16	8 words describing crash codes
47	sbUserName	12	name of logged on user
59	sbOsVersion	64 or less	operating system version
Record Type: System Boot			
31	sbOsVersion	13	operating system version
(continued)			

(continued)

Log File Record Format

Offset	Field	Size (bytes)	Description
Record Type: ISAM Error			
31	erc	2	status code
33	ercDetail	2	detail status code
35	logKind	1	kind of log entry
36	procFrom	1	procedure from
37	uri	4	universal record identifier
41	zbDataSetName	94	data set name
Record Type: Message Text			
31	cbText	2	length of text
33	text	132	text
Record Type: Configuration Error			
31	zbAreaName	78	configuration area name
109	zbDescription	79	description of the error
Record Type: Disk Error (SCSI Logical Block Address Devices)			
31	erc	2	status code
33	retryCnt	1	retry count
34	fRecovered	1	recovered
35	deviceClass	1	device class
36	rgUnitNum	4	unit information
40	command	13	command
53	fLba	1	LBA type device
54	lba	4	logical block address
58	nSectors	1	number of sectors
59	status	20	SCSI status
79	sbVolName	13	volume name

(continued)

Log File Record Format

(continued)

Offset	Field	Size (bytes)	Description
Record Type: Disk Error (Winchester/SMD Devices)			
31	erc	2	status code
33	retryCnt	1	retry count
34	fRecovered	1	recovered
35	deviceClass	1	device class
36	rgUnitNum	4	unit information
40	command	13	command
53	fLba	1	LBA type device
54	cylinder	2	cylinder number
56	head	1	head number
57	sector	1	sector number
58	status	20	drive status
78	sbVolName	13	volume name
Record Type: Error Correction Code (ECC) Error			
31	nSingleBitErrors	2	number of single bit errors
33	pollInterval	2	poll interval in seconds
35	nSyndromeLogged	2	number of logged ECC errors
37	log	8-64	ECC log (<i>nSyndromeLogged</i> * 8)
Record Type: Operating System Error			
31	typeOSerc	2	error type
33	status	8	status information
41	spec	varies	file/device specification

Log File Header Field Descriptions

Following are descriptions of each field in the Log File record header.

sRecord

indicates the size of the log record, which includes the size of *sRecord*.

dateTime

are the date and time that the record was written to the log. *dateTime* is in system date/time format (See "System Date/Time Structure" in this chapter.)

logType

is the type of record. The values of *logType* are described below. (For a description the fields in each record type, see "Type-Specific Field Descriptions," following "Log File Header Field Descriptions."):

Value	Type	Description
3189	ISAM	Logged by the ISAM Service.
3190	ISAM	Logged by the ISAM Service.
3191	ISAM	Logged by the ISAM Service.
65522	Operating System Error	Logged by the operating system.
65523	ECC Error	Logged by the operating system.
65524	ECC Poll Suspend	Logged by the operating system.
65525	Disk Error	Logged by the file system.
65526	Tape Error	Logged by the Half-Inch and Quarter-Inch Tape Services.

Log File Record Format

(continued)

Value	Type	Description
65527	Message Text	Logged by a system service or an application by issuing the WriteLog request.
65528	Configuration	Logged by the operating system.
65529	Buffer Overflow	Logged by the operating system. This record consists of only the Log File header (first 30 bytes of Log File format).
65532	Disk Error (pre-3.0)	Logged by the file system.
65533	System Init Error	Logged by the operating system.
65534	System Boot	Logged by the operating system. This is an information message only.
65535	System Crash	Logged by the operating system.

hardwareType

is the type of hardware upon which the logged event occurred. For the hardware type values, see the *hardwareType* field description in "System Configuration Block" in this chapter.

clusterConfig

is the type of cluster configuration. For the configuration type values, see the *clusterConfig* field description in "System Configuration Block" in this chapter.

fNoFilesystem

is a flag that indicates whether or not the machine causing the record to be logged has a local file system. *fNoFilesystem* is FALSE if there is a local file system.

fCommIop

no longer used.

bPartitionType

is the type of memory management for the partition. For the partition type values, see the description of the field *bPartitionType* in "System Configuration Block" in this chapter.

processorType

is the type of processor on a workstation that caused the record to be logged. For the processor type values, see the description of the field *processorType* in "Port Structure" in this chapter.

bMfpId

is the identification number of the Master File Processor (shared resource processors only).

bMySlot

is the slot number of the processor board that caused the record to be logged (shared resource processors only).

lineNum

is the low-order byte of the word *statParam* in the GetClusterStatus operation. (See the description of GetClusterStatus for details.)

wsId

is the high-order byte of the word *statParam* in the GetClusterStatus operation. (See GetClusterStatus for details.)

cSeg

is the size of memory (in segments) on the processor that caused the record to be logged.

sbUserName

is the name of the user who was logged onto the workstation at the time the record was logged. (The first byte of *sbUserName* is the length of the name.)

typeSpecInfo

is a variable length field containing information specific to this record.

Type-Specific Field Descriptions

Following are the type-specific field descriptions for each record type described in the Log File record header field *logType*.

Record Type: System Initialization Error

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

localErc

is the local status code indicating the type of error that a Shared resource processor operating system encountered during initialization. *localErc* is valid only if the field *erc* has the value 106.

lineNrOrProcType

is the number of the errant line in the configuration file if *erc* is 106. If *erc* is 153 *lineNrOrProcType* is the board type of the processor that encountered the error. (See the values of the *hardwareType* field in the record header.)

text

is text.

Record Type: Disk Error (pre-3.0 Operating Systems Only)

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

retryCnt

is the number of times the command was tried.

fRecovered

is a flag that is TRUE if last retry was successful.

deviceClass

is the class of disk drive caused the record to be logged. (See the field *deviceClass* in "Device Control Block" in this chapter.)

unitNum

is the drive number of the disk that caused the record to be logged.

command

is the command (for example, read, write, or seek) that the drive was executing when the event occurred.

status

is the status of the drive when the event occurred. (See the hardware manual for the disk in question.)

sbVolName

is the name of the volume name of the disk in question. (The first byte of *sbVolName* is the length of the name.)

scsiStatus

is the status of the drive when the event occurred. *ScsiStatus* is valid only if the disk is a SCSI disk. (See the hardware manual for the disk in question.)

Record Type: Tape Error

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

retryCnt

is the number of times the command was tried.

fRecovered

is a flag that is TRUE if last retry was successful.

deviceClass

is the class of tape drive caused the record to be logged. *deviceClass* can be one of the following values:

Value	Class
0	Half-Inch
1	Quarter-Inch Cartridge

unitNum

is the drive number of the disk that caused the record to be logged.

status

is the status of the drive when the event occurred. (See the hardware manual for the disk in question.)

Record Type System Crash

crashCodes

are 8 words that describe the state of the processor at the time of a system crash. (See the *CTOS Status Codes Reference Manual*. The crash words are also described in detail in the *CTOS Debugger User's Guide*.)

sbUserName

is the name of the user who was logged on at the time the record was logged. (The first byte of *sbUserName* is the length of the name).

sbOsVersion

is the version of the operating system that the processor was executing at the time of the event. (The first byte of *sbOsVersion* is the length of the version string).

Record Type System Boot

sbOsVersion

is the version of the operating system that the processor was booting. (The first byte of *sbOsVersion* is the length of the version string).

Record Type ISAM Error (for ISAM types 3189, 3190, and 3191.)

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

ercDetail

is the ISAM status code. (See the *CTOS Indexed Sequential Access Method (ISAM II) Programming Reference Manual*.)

logKind

is the kind of operation that caused the error (used internally only).
The values of *logKind* are

Value	Operation
0	Allocation
1	Deallocation
2	Read
3	Write

procFrom

is the procedure that caused the error (used internally only). The
values of *procFrom* are

Value	Procedure
0	Modify
1	Backout Modification
2	Store
3	Allocate
4	DeAllocate

uri

is the universal record identifier. (See the *CTOS Indexed Sequential Access Method (ISAM II) Programming Reference Manual*.)

zbDataSetName

is the name of the data set against which the error was logged.
zbDataSetName is null terminated. The combined length of the data set
name and the null terminator cannot exceed 94 bytes.

Record Type Message Text

cbText

is the size of the text (in bytes.)

text

is the text of the message that the application wrote to the log.

Record Type Configuration Error

zbAreaName

is the name of the area in the configuration file (for example, EnterDebuggerOnFault) that caused the error. *zbAreaName* is a null terminated string. The combined length of the area name and the null terminator cannot exceed 78 bytes.

zbDescription

is the description of the error. *zbDescription* is a null terminated string. The combined length of the description and the null terminator cannot exceed 79 bytes.

Record Type: Disk Error (SCSI Logical Block Address Devices)

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

retryCnt

is the number of times the command was tried.

fRecovered

is a flag that is TRUE if last retry was successful.

deviceClass

is the class of disk drive caused the record to be logged. (See the field *deviceClass* in "Device Control Block" in this chapter.)

rgUnitNum

describes the device that caused the error to be logged as follows:

Offset	Size	Description
0	1	Reserved
1	1	SCSI bus number
2	1	Target ID
3	1	LUN

command

is the command (for example, read, write, seek, and so forth) that the drive was executing when the event occurred.

fLBA

is a flag that is TRUE to indicate that the device is logically blocked.

LBA

is the logical block address of the sector that caused the error.

(continued)

Log File Record Format

nSectors

is the number of sectors involved in the error.

status

is the status of the drive when the event occurred. (See the hardware manual for the disk in question.)

sbVolName

is the name of the volume name of the disk in question. (The first byte of *sbVolName* is the length of the name.)

Record Type: Disk Error (Winchester/SMD Devices)

erc

is the status code. (See the *CTOS Status Codes Reference Manual*.)

retryCnt

is the number of times the command was tried.

fRecovered

is a flag that is TRUE if last retry was successful.

deviceClass

is the class of disk drive caused the record to be logged. (See the field *deviceClass* in "Device Control Block" in this chapter.)

rgUnitNum

describes the device that caused the error to be logged as follows:

Offset	Size	Description
0	1	Reserved
1	1	Controller number (Winchester Only) Unit number (SMD Only)
2	1	Unit number (Winchester Only)
3	1	Reserved

command

is the command (for example, read, write, seek, and so forth) that the drive was executing when the event occurred.

fLBA

is a flag that is FALSE to indicate that the device is not logically blocked.

cylinder

is the cylinder number of the sector that caused the error.

head

is the head number of the sector that caused the error.

sector

is the sector number of the sector that caused the error.

status

is the status of the drive when the event occurred. (See the hardware manual for the disk in question.)

(continued)

Log File Record Format

sbVolName

is the name of the volume name of the disk in question. (The first byte of *sbVolName* is the length of the name.

Record Type: Error Correction Code (ECC) Error

nSingleBitErrors

total number of single bit errors logged since the last poll interval.

pollInterval

last interval in seconds between reporting of ECC errors.

nSyndromeLogged

number of different ECC error types logged in this entry.

log

array of (8 byte) logs for each different ECC error type logged.

Record Type: Operating System Error

typeOsErc

is the operating system internal error type.

status

is the status information associated with each internal error, consisting of one to four entries, each two bytes long, of the following information in any order:

the status code that caused the condition

the class of the operating system error

a line number in a configuration file which is in error

a file header number

a file system bit-map page number

a port on a processor board

spec

One or two file or device specifications. The length of the specification information is determined by subtracting the offset of the field *spec* from the field *sRecord* in the header.

Description

Physical addresses 0 to 3FFh in the *low memory allocation* table that follows are the addresses of interrupt routines in low memory on real mode operating systems. If, for example, your real mode program needed to find the Debugger breakpoint procedure, the memory address of this code can be found at offset 0Ch. (In real mode on protected mode operating systems, the addresses actually point to code that switches to protected mode to execute interrupts.) In the table you will note that specified address ranges are available for applications to use as traps.

Because all interrupts in protected mode are handled through the Interrupt Descriptor Table, the physical addresses in the table have no meaning. However, the table does provide a description of each interrupt by its interrupt number. This information is true for protected mode as well as real mode systems. (See the table columns, *Interrupt Number* and *Description*.) In protected mode, interrupt numbers are in the IDT and can only be accessed through the INT instruction or by hardware interrupts. They cannot be accessed programmatically. Interrupt handlers can be established using the SetIntHandler or SetDeviceHandler operation.

Starting at physical address 400h (*XBIS* field), three fields are true low memory physical addresses on both real and protected mode systems.

Related Timer and Interrupt Handler Structures

- Expanded Date/Time Format
- System Date/Time Structure
- Timer Pseudointerrupt Block
- Timer Request Block Format

This page intentionally left blank

(continued)

Low Memory Allocation

Physical Address	Size	Field/Interrupt Number	Description
0h	4h	Int 00h	Divide error trap.
04h	4h	Int 01h	Single-step trap (used by debugger).
08h	4h	Int 02h	Nonmaskable Interrupts (NMI) (memory or bus timeouts, parity errors).
0Ch	4h	Int 03h	Breakpoint trap (used by the Debugger).
10h	4h	Int 04h	Signed arithmetic overflow trap.
14h	4h	Int 05h	Array bounds exception trap
18h	4h	Int 06h	Unused opcode exception
1Ch	4h	Int 07h	ESC opcode exception
20h	60h	Int 08h -Int 1Fh	Hardware interrupt vectors. (See hardware manuals for various processors.)
80h	32h	Int 20h Int 28h	Software trap vectors available for applications
B2h	8	Int 28h Int 29h	Used by PMOS
BAh	46h	Int 2Ah-Int 3Fh	Software trap vectors available for applications
100h	20h	Int 40h-Int 47h	Hardware interrupt vectors. (See hardware manuals for various processors.)
120h	4h	Int 48h	Kernel entry vector
124h	4h	Int 49h	System-common entry vector
128h	4h	Int 4Ah	Procedural interface entry vector
(continued)			

Low Memory Allocation

(continued)

Physical Address	Size	Field/Interrupt Number	Description
12Ch	4h	Int 4Bh	Floating point exception.
130h	10h	Int 4Ch-Int 4Fh	Reserved for future hardware and operating system interrupt and trap vectors.
140h	20h	Int 50h-Int 57h	Hardware interrupt vectors. (See hardware manuals for various processors.)
160h	70h	Int 58h-Int 73h	Reserved for future hardware and operating system interrupt and trap vectors.
1D0h	10h	TapeBlock	Data area reserved for quarter-inch and half-inch tape software and system services.
1E0h	10h	Int 78h-Int 7Bh	Reserved for future hardware and operating system interrupt and trap vectors.
1F0h	10h	BootArea	Data area set up by boot ROM for communicating with the operating system (real mode only).
200h	10h	rgCEntryInt	Array of software trap INT instructions for Kernel primitives (real mode only).
210h	10h	rgSysComInt	Array of software trap INT instructions for system-common procedures (real mode only).
220h	10h	rgRqInflnt	Array of software trap INT instructions for request procedural interfaces (real mode only).
230h	10h	Int 8Ch-Int 8Fh	Reserved for future hardware and operating system interrupt and trap vectors.

(continued)

(continued)

Low Memory Allocation

Physical Address	Size	Field/Interrupt Number	Description
240h	0B0h	rgpSysCom	Array of addresses previously called the System Common Address Table (SCAT) (real mode only). While still supported, all applications (real and protected mode) should use GetPStructure, which describes these addresses.
2F0h	10h	Int 0BCh-Int 0BFh	Reserved for future hardware and operating system interrupt and trap vectors.
300h	30h	Int 0C0h-Int 0CCh	Software trap vectors available for applications.
334h	4h	Int 0CDh	Protected mode procedural interface.
338h	C8h	Int 0CEh-Int 0FFh	Software trap vectors available for applications.
400h	200h	XBIS*	X-bus interface structure.
600h	200h	286State*	State information used with 80286 processors when switching between protected mode and real mode.
800h	200h	LoadAllData*	Internal 80286 registers used when executing LOADALL instruction (used by the Debugger).

*These fields are supported on all operating systems.

This page intentionally left blank

Description

When a disk volume is initialized, the volume control structures for that volume are created. The volume control structures include the following:

- Volume Home Block
- File Header Block
- Master File Directory
- directories
- Allocation Bit Map

The volume control structures allow the file management system to manage the space on a volume not dedicated to the control structures themselves. (For details on all the volume control structures and their realationships, see "File Management" in the *CTOS Operating System Concepts Manual*.)

The maximum number of directories that can be created on a volume depends on the number of *Master File Directory (MFD) blocks* specified at volume initialization. The structure that follows shows the format of each MFD block. Note that hashing techniques allow up to a maximum of 14 directory entries in a single MFD block.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Directory Entry in a Master File Directory Block
- Entry for a File in a Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

(continued)

Master File Directory Block

Offset	Field	Size (bytes)	Description
0	fOverflow	1	indicates if the MFD block is full
1	cbDirEntries	$n * 35$	total byte count of directory entries
$1+n * 35$	nullTerminator	1	terminating character of a MFD block

fOverflow

is a flag. If *fOverflow* is TRUE, the MFD block is full or once was full, causing one or more directory entries that hashed to this block to be placed in the next block instead.

cbDirEntries

contains the byte count for directory entries in this MFD block. Each directory entry is 35 bytes, and the number of directory entries (n) in one MFD block cannot exceed 14. (For the format of a directory entry, see "Entry for a Directory in a Master File Directory Block" in this chapter.)

nullTerminator

is the last character a MFD block.

This page intentionally left blank

Multibyte String Masks Table

Description

The *Multibyte String Masks Table* identifies whether a key combined with the current chord state will generate a multibyte string.

One entry exists for each key. Each entry in the Multibyte String Masks Table corresponds to an entry in the emulation or translation tables. The position of the set bit in a Multibyte String Masks Table entry identifies the emulation or translation table in which the key generates a string.

For example, if the 5th bit in the second Multibyte String Masks Table entry is set, the keyboard process determines that the second key in the 5th emulation or translation table generates a string.

A Multibyte String Masks Table resides in a translation or emulation data block. The *oStringMasks* field of the Translation or Emulation Data Block Header contains the offset, from the beginning of the data block, of the Multibyte String Masks Table. The *cKeys* field of either header contains the number of entries in the Multibyte String Masks Table.

The Multibyte String Masks Table only exists if the Multibyte Strings Table is present. If no Multibyte String Masks Table exists, *oStringMasks* is 0.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Repeat Attributes Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Multibyte String Masks Table

Offset	Field*	Size (bytes)	Description
0	stringMaskEntry1	2	mask value for first table entry
2	stringMaskEntry2	2	mask value for second table entry
.			
.			
.			

*A Multibyte String Masks Table contains a variable number of entries.

stringMaskEntry1

is a mask value that identifies the emulation or translation table in which the first entry references a string.

stringMaskEntry2

is a mask value that identifies the emulation or translation table in which the second entry references a string.

.

.

.

and so forth.

This page intentionally left blank.

Multibyte String Offsets Table

Description

The *Multibyte String Offsets Table* is used by the keyboard process to identify the offset, from the beginning of the data block, of each entry in the Multibyte Strings Table. (See the Multibyte Strings Table.)

The first entry in the Multibyte String Offsets Table is always 0. It is intended only for internal use. The other entries contain the data block offsets of entries in the Multibyte Strings Table.

A Multibyte String Offsets Table can reside in either an emulation data block or a translation data block. The data block offset of the Multibyte String Offsets Table is located in the *oStringOffsets* field of the Translation or Emulation Data Block Header. The number of entries in the Multibyte String Offsets Table is located in the *cStringOffsets* field of either header.

A multibyte string offsets table only exists if a strings table is present in the data block. If a multibyte string offsets table is not present, the *oStringOffsets* field of the Translation or Emulation Data Block Header is 0.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Multibyte String Masks Table
- Multibyte Strings Table
- Repeat Attributes Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Multibyte String Offsets Table

Offset	Field*	Size (bytes)	Description
0	nullValue	2	the number 0
2	oEntry1	2	offset of first entry in Multibyte Strings Table
4	oEntry2	2	offset of second entry in Multibyte Strings Table
.			
.			
.			

*Multibyte String Offsets Tables contain a variable number of entries.

nullValue

is always 0.

oEntry1

is the offset of the first entry in the Multibyte Strings Table.

oEntry2

is the offset of the second entry in the Multibyte Strings Table.

.

.

.

and so forth.

This page intentionally left blank.

Description

The *Multibyte Strings Table* is a keyboard emulation or translation support table that contains an array of string entries. Each entry is indexed by the raw or emulated key post value of the key that will generate the string when pressed.

Initially, the keyboard process uses the Multibyte String Masks Table to determine whether the emulation or translation table entry for a key is an index into the Multibyte Strings Table. If the entry is an index value, the keyboard process subsequently retrieves the offset to the string header from the Multibyte String Offsets Table. Then, using the offset, the appropriate string is retrieved from the Multibyte Strings Table.

During data block construction, the end user should include an equal number of upstroke values and downstroke values in each string. If upstroke values are missing, the Keyboard Customization Tool automatically appends them to the string.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Repeat Attributes Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Multibyte Strings Table

Offset	Field*	Size (bytes)	Description
varies	string1	varies	string produced by key 1
.			
.			
varies	string255	varies	string produced by key 255

*Multibyte Strings Tables contain from 1 to 256 entries.

string1

is the string generated by a keyboard or character code of 0. All entries in the Multibyte Strings Table have the following format:

Field	Size (bytes)	Description
cbUserSupplied	2	The size, in words, of the string portion supplied by the user.
cbCleanup	2	The size, in words, of the string portion added by the Keyboard Customization Tool.
userSupplied	varies	The portion of the string supplied by the user.
cleanup	varies	The portion of the string added by the Keyboard Customization Tool.

Multibyte Strings Table

(continued)

string2

is the string generated by a keyboard or character code of 1.

.

.

.

maximum number of strings

is the string generated by a keyboard or character code of 255. This entry has the same format as the first entry.

Description

The operating system flags structure uses flags to indicate the type of hardware upon which a program is running. These flags are set at initialization. Thereafter, a program can refer to the flags rather than interrogating the actual hardware.

This page intentionally left blank.

(continued)

Operating System Flags

Offset	Field	Size (bytes)	Description
0	f186	1	186 based processor
1	f286	1	286 based processor
2	f386	1	386 based processor
3	fXBus	1	X-Bus is connected to this processor
4	fNewGen	1	TRUE if an integrated workstation
5	fMP	1	set for CTOS/XE
6	fMasterFp	1	master processor
7	fPaging	1	386 with paging enabled
8	fMaster	1	RS-422 cable is connected
9	fClstr	1	cluster workstation
10	fFS	1	file system is on this processor
11	f287	1	287 floating point chip is present
12	f387	1	387 floating point chip is present
13	f386Tss	1	supports 32-bit addressing
14	fPS2	1	set for SG5000 or EISA/ISA-bus workstation
15	fTapeBoot	1	correct OS and [Sys] is memory disk
16	fOsLoader	1	first phase of bootstrapping from tape
17	fLptIn	1	LPT hardware can receive input
18	fPageServiceInstalled	1	Page Service is installed
19	fVSeries	1	virtual memory operating system or greater
20	fExp3Port	1	FAS comm port is available
21	f486	1	system has a 486 CPU
22	f4Mbit	1	system is capable of 4megabit cluster speed
23	fB38LCW	1	system is a B38Lcw
24	ffsGen	1	system is a SG5000
25	fXBusVideo	1	system is capable of having video on Xbus
26	fpcAT	1	system is a EISA/ISA-bus workstation
27	fStatistics	1	statistics server is present
28	fPortable	1	system is a laptop workstation

f186

is set to TRUE if this is a 186 based processor.

f286

is set to TRUE if this is a 286 based processor.

f386

is set to TRUE if this is a 386 based processor.

fXBus

is set to TRUE if there an X-Bus is connected to this processor.

fNewGen

is set to TRUE if this is an integrated (Series 286i, or 386i, or a B39) workstation.

fMf

is set to TRUE if this is a CTOS/XE shared resource processor.

fMasterFp

is set to TRUE if this is the master processor of a shared resource processor.

fPaging

is set to TRUE if this is a 386 based processor with paging enabled.

fMaster

is set to TRUE if an RS-422 cable is connected to this processor.

fClstr

is set to TRUE if this is a cluster workstation.

fFS

is set to TRUE if a file system is present on this processor.

f287

is set to TRUE if a 287 floating point chip is present.

f387

is set to TRUE if a 387 floating point chip is present.

f386Tss

is set to TRUE if this processor supports 32-bit addressing.

fPS2

is set to TRUE if the system is an SG5000 or a EISA/ISA-bus workstation.

fTapeBoot

is set to TRUE if this is the second phase of bootstrapping from tape (*fOsLoader* is FALSE), and [Sys] is a memory disk.

fOsLoader

is set to TRUE if this is the first phase of bootstrapping from tape.

fLptIn

is set to TRUE if LPT hardware can receive input.

fPageServiceInstalled

is set to TRUE if page service is installed.

Operating System Flags

(continued)

fVSeries

is set to TRUE if the operating system is CTOS III or greater.

fExp3Port

is set to TRUE if FAS communications port is available.

f486

is set to TRUE if system has a 486 CPU.

f4Mbit

is set to TRUE if system is capable of 4 megabit cluster speed.

fB38LCW

is set to TRUE if system is a B38Lcw.

fSGen

is set to TRUE if system is an SG5000.

fXBusVideo

is set to TRUE if system is capable of having video on X-Bus.

fpcAT

is set to TRUE if system is a EISA/ISA-Bus workstation.

fStatistics

is set to TRUE if the statistics server is present.

fPortable

is set to TRUE if the system is a laptop workstation.

Description

The *Partition Configuration Block* contains the addresses of the following application partition structures:

- Extended Partition Descriptor
- Batch Control Block
- Application System Control Block
- Extended User Structure

Each of these structures are part of the User Structure for an application partition. For additional information on the User Structure, see "Partitions and Partition Management" in the *CTOS Operating System Concepts Manual*. The formats of the first three structures listed above are presented in this chapter.

Related Partition Structures

Application System Control Block
Batch Control Block
Extended Partition Descriptor
Partition Descriptor
Variable Length Parameter Block

This page intentionally left blank

(continued)

Partition Configuration Block

Offset	Field	Size (bytes)	Description
0	<i>oExtendedParDesc</i>	2	address of Extended Partition Descriptor
2	<i>oBCB</i>	2	address of Batch Control Block
4	<i>oASCB</i>	2	address of ASCB
6	<i>oUSegDesc</i>	2	address of extended User Structure

oExtendedParDesc

is the address of the Extended Partition Descriptor. See "Extended Partition Descriptor" in this chapter.

oBCB

is the address of the Batch Control Block. See "Batch Control Block" in this chapter.

oASCB

is the address of the Application System Control Block. See "Application System Control Block" in this chapter.

oUSegDesc

is the address of an area describing additional user structure segment(s) in the extended User Structure. A program obtains memory dynamically from this area by calling the `ReservePartitionMemory` operation. (For details, see the description of `ReservePartitionMemory` in Chapter 3, "Operations.")

This page intentionally left blank

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

The *Partition Descriptor* contains the partition name, the boundaries of the partition and of its long- and short-lived memory areas, and internal links to Partition Descriptors in other partitions. Real mode operating systems access the User Structure of a partition through the Partition Descriptor. Access by other programs, however, is usually done by calling the GetPStructure operation. (For details, see "Partitions and Partition Management" in the *CTOS Operating System Concepts Manual*.)

With the exception of the field sbPartitionName, this structure is not used by protected mode operating systems. Most of the information it contains is available through other internal structures.

Related Partition Structures

- Application System Control Block
- Batch Control Block
- Extended Partition Descriptor
- Partition Configuration Block
- Variable Length Parameter Block

This page intentionally left blank

(continued)

Partition Descriptor

Offset	Field	Size (bytes)	Description
0	<i>oForwardLink</i>	2	offset to next Partition Descriptor
2	<i>oBackwardLink</i>	2	offset to previous Partition Descriptor
4	<i>saLowBound</i>	2	address of User Structure
6	<i>saMinLL</i>	2	address of lower end of LL memory
8	<i>saCurLL</i>	2	address of higher end LL memory
10	<i>saCurSL</i>	2	address of lower end of SL memory
12	<i>saMaxSL</i>	2	address of higher end of SL memory
14	<i>saHighBound</i>	2	address of high end of partition
16	<i>sbPartitionName</i>	13	name of partition
29	<i>fPartitionVacant</i>	1	indicates if partition is vacant
30	<i>fPartitionLocked</i>	1	indicates if partition is locked
31	<i>PartitionExchange</i>	2	is the partition exchange
33	<i>fBatchPartition</i>	1	Batch Control Block allocated or not
34	<i>fCCB</i>	1	partition under context mangement
35	<i>cParCreatedMSB</i>	1	high 8 bits of <i>cParCreated</i>
36	<i>cParCreated</i>	2	paragraphs count when partition created
38	<i>saMinCode</i>	2	address at which code starts
40	<i>saMaxCode</i>	2	address at which code stops

oForwardLink

is the offset to next Partition Descriptor.

oBackwardLink

is the offset to previous Partition Descriptor.

saLowBound

is the segment base address of the User Structure. The User Structure consists of the partition structures previously listed in "Related Partition Structures."

saMinLL

is the segment base address of the lower end of long-lived memory.

saCurLL

is the segment base address of the higher end of long-lived memory (lower end of free memory).

saCurSL

is the segment base address of the lower end of short-lived memory (higher end of free memory).

saMaxSL

is the segment base address of the higher end of short-lived memory.

saHighBound

is the segment base address of the high end of the partition.

sbPartitionName

is the name of the partition. The first byte contains the size of the name string.

fPartitionVacant

is a flag that is TRUE if the partition is vacant.

fPartitionLocked

is a flag that is TRUE if the partition is locked.

PartitionExchange

is the partition exchange.

fBatchPartition

is a flag that is TRUE if a Batch Control Block is allocated.

fCCB

is a flag that is TRUE if the partition is under the management of a context managing program, such as the Context Manager.

cParCreatedMSB

is the high 8 bits of *cParCreated*. (See below.)

cParCreated

is the count of paragraphs in the partition when the partition was first created.

saMinCode

is the segment base address of the first byte of code.

saMaxCode

is the segment base address of the last byte of code.

This page intentionally left blank

Description

The *Partition Information Block* contains information about the partition of a given user number. Data in this structure includes the selector of the partition's Partition Configuration Block, the table (GDT or LDT) referenced by the partition's owner, and the maximum number of 16-byte paragraphs allocated to the partition.

An application can obtain the Partition Information Block by calling `GetPartitionStatus` with a *StatusCode* value of 5.

Related Partition Structures

- Application System Control Block
- Batch Control Block
- Extended Partition Descriptor
- Partition Configuration Block
- Partition Descriptor

This page intentionally left blank

(continued)

Partition Information Block

Offset	Field	Size (bytes)	Description
0	cbRet	2	count of bytes returned in this structure
2	userNum	2	user number of partition owner
4	descTable	2	descriptor table referenced by partition
6	cParMax	4	highest number of paragraphs for partition
10	sgUser	2	selector of Partition Configuration Block

cbRet

is the count of bytes in the Partition Information Block.

userNum

is the user number of the partition owner.

descTable

identifies the descriptor table that is referenced by the partition's owner. Each value indicates the following:

Value	Description
0	Global Descriptor Table
1	No table (real mode)
2	Local Descriptor Table

cParMax

is the maximum number of 16-byte paragraphs allocated to the partition.

sgUser

is the selector of the partition's Partition Configuration Block.

This page intentionally left blank

Partition Keyboard Information Structure

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

The *Partition Keyboard Information Structure* is used by the keyboard process to record keyboard information for a given user number. Data in this structure includes the addresses of the translation and emulation data blocks for the user number, the characteristics of the last diacritic key pressed, and the current state of the keyboard LEDs.

When the user switches contexts, the keyboard process saves the current keyboard state of the user number in the Partition Keyboard Information Structure. When the application resumes execution, the keyboard process reads this structure to identify keys that were pressed or toggled before the context switch occurred. It then restores the original state of the keyboard.

As an example, suppose a particular keyboard LED is activated in one context and turned off in another context. When the user switches to a new context, the keyboard process reads this structure and accordingly turns the LED on or off.

For details on keyboard management, see "Keyboard and I-Bus Management" in the *CTOS Operating System Concepts Manual*.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Partition Keyboard Information Structure

Offset	Field	Size (bytes)	Description
0	pKbdTransHeader	4	address of Translation Data Block Header
4	pKbdEmulHeader	4	address of Emulation Data Block Header
8	postEmulState	2	state of chord after emulation
10	keyState	2	state of chords from type-ahead buffer
12	trueChordState	2	state of chord key from interrupt buffer
14	pRepeatBuffer	4	pointer to buffer that holds repeat key data
18	bitMapDownstrokes	16	bit map of down keys from type-ahead
34	fSemicoded	1	TRUE indicates semicoded mode
35	keyboardID	1	ID of currently attached keyboard
36	iDiaTable	1	index of translation table for diacritic key
37	diaChordState	2	chord state for last diacritic key pressed
39	diaKeyCode	1	keyboard code of last diacritic key pressed
40	keyCodeAfterDiac	1	keyboard code of key after diacritic key
41	cExpansionStrings	1	count of expansion strings in use
42	rgwExp	6	array that maps keys to string numbers
48	repeatAttr	1	repeat attributes for last key that repeated.
49	dbgrLedState	2	state of the LEDs when in the Debugger
51	highLedState	1	state of additional keyboard LEDs
52	operatingSystemFlags	1	bit masks that describe user mode
53	asiaLedState	1	reserved
54	partitionNumber	2	partition number
56	wAsiaRepeat	2	reserved

Partition Keyboard Information Structure

(continued)

Offset	Field	Size (bytes)	Description
58	lastKey	1	last key read from interrupt buffer
59	fAsiaCodeMode	1	reserved
60	pAsiaBuffer	4	reserved
64	iBlocks	1	index into array of OS tables
65	reserved	15	reserved

pKbdTransHeader

is the memory address of the user number's Translation Data Block Header. This value is changed by the PostKbdTable operation when an application specifies a new translation data block. (See Chapter 3, "Operations," for a description of PostKbdTable.)

pKbdEmulHeader

is the memory address of the user number's Emulation Data Block Header. This value is changed by PostKbdTable when an application specifies a new emulation data block.

postEmulState

is the state of the chord keys after emulation.

keyState

is the state of chord keys read from the type-ahead buffer.

trueChordState

is the state of the chord keys read from the interrupt buffer.

pRepeatBuffer

is the memory address of a buffer that holds repeat key data.

bitMapDownstrokes

is the bit map of downstrokes read from the type-ahead buffer.

fSemicoded

is a flag. TRUE indicates that the user number has requested semicoded mode.

keyboardID

is the ID of the currently attached keyboard.

iDiaTable

is the index number of the translation table in which the last key pressed is defined as a diacritic key.

diaChordState

is the chord state of the last diacritic key pressed.

diaKeyCode

is the keyboard code of the last diacritic key pressed.

keyCodeAfterDiac

is the keyboard code of the key that is pressed after the diacritic key.

cExpansionStrings

is the count of expansion strings currently in use by the user number.

rgwExp

is an array that maps keys to string numbers.

repeatAttr

is a byte that defines attributes for the last key that repeated.

dbgrLedState

is the state of the keyboard LEDs when in the Debugger.

highLedState

is the state of the keyboard LEDs not mentioned in the Extended Partition Descriptor. Examples include communication LEDs and Asian keyboard LEDs.

operatingSystemFlags

is a byte containing several flags. Certain bit positions, when set or cleared, indicate the following:

Bit Position	Value	Description
0	0	Do not provide the user number with keyboard codes.
	1	Provide the user number with keyboard codes.
4	0	Do not disable direct input from the keyboard and mouse.
	1	Disable direct input from the keyboard and mouse.
5	0	The user number is in single-key repeat mode.
	1	The user number is not in single-key repeat mode.
6	0	Use the OS default emulation table.
	1	Use the emulation table posted by the user number.

(continued)

Partition Keyboard Information Structure

Bit Position	Value	Description
7	0	Use the OS default translation table.
	1	Use the translation table posted by the user number.

asiaLedState

is reserved for use by Asian versions of the keyboard process.

partitionNumber

is the partition number.

wAsiaRepeat

is reserved for use by Asian versions of the keyboard process.

lastKey

is the most recent key read from the interrupt buffer.

fAsiaCodeMode

is reserved for use by Asian versions of the keyboard process.

pAsiaBuffer

is reserved for use by Asian versions of the keyboard process.

iBlocks

is an index into an array of OS tables.

reserved

is reserved for future use.

This page intentionally left blank.

Description

The *Performance Statistics Structure* is the format that clients of the Performance Statistics system service use to collect statistics about the system. The areas of statistics that the client can query include CPU-related information such as task switches and swapped partitions, disk errors, files, read/write seeks, and read/write accesses. (The Performance Statistics system service is described in detail in the *CTOS Programming Guide*. To determine which operating systems support this service, see Appendix A, "Operating System Features," in the *CTOS Operating System Concepts Manual*.)

Each area of statistics is structured as a block. Each block is identified by a number, starting with block number 1. Except for block number 1, which describes CPU-related information, all other blocks contain statistics on devices connected to the X-Bus. The index tells the relative position of the device along the X-Bus. Statistics about a device can be collected for up to a maximum of 18 devices connected to the CPU.

To open a statistics collecting session, the client calls `PSOpenStatSession` and specifies the information it wants to collect for given blocks. (See the description of *pbBlockDesc/cbBlockDesc* in the `PSOpenStatSession` operation.)

After opening a statistics collecting session, the client can call `PSGetCounters` to collect statistics and `PSResetCounters` to reset the counters.

Related Structures

None

This page intentionally left blank

(continued)

Performance Statistics Structure

Block Number	Index	Offset	Counts the Number of
1	0	0	Idle process cycles
	0	4	Normal task switches
	0	8	Processes terminated normally
	0	12	Processes terminated abnormally
	0	16	Partitions swapped into memory
	0	20	Partitions swapped out of memory
	0	24	
2	0	0	Hard Disk Errors
	0	4	Soft Disk Errors
	0	8	IOB's in process
	1	0	Hard Disk Errors
	1	4	Soft Disk Errors
	1	8	IOB's in process
.	.	.	
.	.	.	
.	.	.	
	17	0	Hard Disk Errors
	17	4	Soft Disk Errors
	17	8	IOB's in process
3	0	0	Files opened
	0	4	Files closed
	0	8	Files created
	0	12	Files deleted
	0	16	Files renamed
	0	20	File length changes
	0	24	Remade file handles
	0	28	
.	.	.	
.	.	.	
.	.	.	
	17	16	Files renamed
	17	20	File length changes
	17	24	Remade file handles
(continued)			

Performance Statistics Structure

(continued)

Block Number	Index	Offset	Counts the Number of
4	0	0	1-sector seeks during Read
	0	4	2-sector seeks during Read
	0	8	3-sector seeks during Read
	.	.	.
	.	.	.
	.	.	.
	0	504	127-sector seeks during Read
	0	508	128-sector seeks during Read
	.	.	.
	.	.	.
	.	.	.
	17	0	1-sector seeks during Read
	17	4	2-sector seeks during Read
	17	8	3-sector seeks during Read
5	.	.	.
	.	.	.
	.	.	.
	17	504	127-sector seeks during Read
	17	508	128-sector seeks during Read
			Same format as block 4 but it counts the number of seeks during Write
6	0	0	Accesses of extents 2^0 bytes long during Read
	0	4	Accesses of extents 2^1 bytes long during Read
	0	8	Accesses of extents 2^3 bytes long during Read
	.	.	.
	.	.	.
	.	.	.
	0	504	Accesses of extents 2^{127} bytes long during Read
	0	508	Accesses of extents 2^{128} bytes long during Read
	.	.	.
	.	.	.
	.	.	.
	.	.	.
	.	.	.
	.	.	.

(continued)

(continued)

Performance Statistics Structure

Block Number	Index	Offset	Counts the Number of
	17	0	Accesses of extents 2^0 bytes long during Read
	17	4	Accesses of extents 2^1 bytes long during Read
	17	8	Accesses of extents 2^3 bytes long during Read
	.	.	.
	.	.	.
	.	.	.
	17	504	Accesses of extents 2^{127} bytes long during Read
	17	508	Accesses of extents 2^{128} bytes long during Read
7			Same format as block 6 but it gives the number of accesses during Write
8*	0	0	1-sector logical Reads
	0	4	2-sector logical Reads
	0	8	3-sector logical Reads
	.	.	.
	.	.	.
	.	.	.
	0	504	127-sector logical Reads
	0	508	128-sector logical Reads
	.	.	.
	.	.	.
	.	.	.
	17	0	1-sector logical Reads
	17	4	2-sector logical Reads
	17	8	3-sector logical Reads
9*			Same format as block 8 but it gives the number of logical Writes

*The appropriate counter is incremented with each Read or Write operation.

This page intentionally left blank

Description

The *Port Structure* contains hardware port addresses of various devices whose memory addresses differ in various configurations.

Each field in the structure is a port address unless noted otherwise in the detailed field descriptions.

This structure can be accessed by calling `GetPStructure` with a *structCode* value of 14.

Related System Structures

Extended Process Control Block

Process Control Block

System Configuration Block

This page intentionally left blank

Port Structure

Offset	Field	Size (bytes)	Description
0	beeperPort	2	enables/disables the speaker
2	ioCommCtlA	2	MPSC command/status, channel A
4	ioCommDataA	2	MPSC data, channel A
6	ioCommCtlB	2	MPSC command/status, channel B
8	ioCommDataB	2	MPSC data, channel B
10	modeWord8253	2	8253/54 programmable counter mode byte
12	baudRateCtrA	2	generates baud rate clock, channel A
14	baudRateCtrB	2	generates baud rate clock, channel B
16	oCW1_8259	2	8259 PIC operation ctrl word output (odd)
18	oCW2_8259	2	8259 PIC operation ctrl word output (even)
20	cascadeOCW1_8259	2	slave operation ctrl word output (odd)
22	cascadeOCW2_8259	2	slave operation ctrl word output (even)
24	extCtlReg	2	secondary transmit data or external clocks
26	earPort	2	maps CPU memory to X-BUS memory
28	parityEnablePort	2	enables/disables memory parity error (NMI)
30	parityErrPort0	2	low 16 bits NMI-latched memory or I/O
32	parityErrPort1	2	high 16 bits NMI-latched memory or I/O
34	vidControl	2	enables/disables video display
36	vid6845Addr	2	selects internal video ctrl data registers
38	vid6845Data	2	writes to internal video ctrl data registers
40	commDmaAddr	2	8237 RS-232 cluster communications
42	commDmaWrldCnt	2	8237 RS-232 cluster word count
44	commDmaEar	2	RS-232 DMA transfer high address bits
46	dmaMask	2	8237 DMA controller channel mask
48	dmaMode	2	8237 DMA controller mode register
50	dmaCommand	2	8237 DMA controller command register
52	dmaBytePtrClr	2	clears first/last 8237 byte flip-flop
54	rdsrL	2	MPCC data port for intracluster comm
56	rdsrH	2	2652 controller receive status
58	tdsrL	2	2652 controller transmit data register
60	tdsrH	2	2652 command and status register
62	pcsarL	2	2652 synchronization/address register
64	pcsarH	2	2652 controller parameter control register
66	pcrL	2	returns zeros when read
68	pcrH	2	2652 controller character length register
70	cPort	2	controls DMA, interrupts, transmit clock
72	stat	2	RS-422 cluster state machine status
(continued)			

Port Structure

(continued)

Offset	Field	Size (bytes)	Description
74	timerCtl	2	8254 counter mode register
76	timerComm	2	8254 counter load register
78	clearRtcIntPort	2	clears pending Realtime Clock interrupt
80	xIntR3vector	2	interrupt vector for level 3 X-Bus interrupts
82	processorType	2	value of workstation processor type
84	dmaEarEnable	2	controls DMA EARs in some processors
86	timeClockBaseAddr	2	first I/O port in Realtime Clock chip
88	nonvolatileRamBaseAddr	2	first I/O port in a RAM block
90	protectedModeEnabl	2	processor configured for protected mode
92	fingerPort	2	processor switch protected mode to real
94	shCpuSpeed	2	processor instruction execution speed
96	videotype	2	value used for video initialization
98	rs232DmaAddr0	2	RS-232 channel a receive portion uses
100	rs232DmaWrdCnt0	2	DMA controller channel 0 word count
102	rs232DmaAddr1	2	DMA controller channel 1 address register
104	rs232DmaWrdCnt1	2	DMA word cnt register for RS-232 transmit
106	rs232DmaAddr2	2	DMA channel 2 address register
108	rs232DmaWrdCnt2	2	DMA word cnt register for RS-232 receive
110	rs232DmaAddr3	2	DMA channel 3 address register
112	rs232DmaWrdCnt3	2	DMA word cnt register for RS-232 transmit
114	rs232DmaCommand	2	DMA command register for all channels
116	rs232DmaRequest	2	DMA controller request register for RS-232
118	rs232DmaMask	2	RS-232 DMA controller individual mask
120	rs232DmaMode	2	RS-232 DMA controller mode register
122	rs232DmaBytePtrClr	2	8237 DMA byte count clear register
124	rs232DmaMstrClr	2	RS-232 DMA master clear register
126	rs232MaskAll	2	RS-232 DMA all channel mask register
128	rs232DmaEar0	2	RS-232 DMA channel 0 EAR
130	rs232DmaEar1	2	RS-232 DMA channel 1 EAR
132	rs232DmaEar2	2	RS-232 DMA channel 2 EAR
134	rs232DmaEar3	2	RS-232 DMA channel 3 EAR
136	rs232DmaEarEnable	2	selects/enables RS-232 EARs
138	ledDisplayPort	2	controls 7-segment LED display

beeperPort

contains bits to enable and disable the workstation speaker and its associated tone circuitry. On some processors, other bits in this port are used to enable/disable the video display and to reset the keyboard. See the appropriate processor hardware manual for details.

ioCommCtlA

is the command and status register for Channel A of the Multi-Protocol Serial Controller (MPSC) (RS-232-C) chip.

ioCommDataA

is the data register for Channel A of the MPSC.

ioCommCtlB

is the command and status register for Channel B of the MPSC.

ioCommDataB

is the data register for Channel B of the MPSC.

modeWord8253

contains the mode byte of the 8253 or 8254 programmable counter that is used to generate the MPSC baud rate clocks.

baudRateCtrA

is the counter register in the 8253 or 8254 programmable counter used to generate the baud rate clock for Channel A of the MPSC.

baudRateCtrB

is the counter register in the 8253 or 8254 programmable counter used to generate the baud rate clock for Channel B of the MPSC.

oCW1_8259

outputs operation control words to the 8259 programmable interrupt controller (PIC)(odd port address).

oCW2_8259

outputs operation control words to the 8259 PIC (even port address).

cascade-OCW1_8259

outputs operation control words to a second or slave 8259 PIC, if one exists (odd port address).

cascade-OCW2_8259

outputs operation control words to a second or slave 8259 PIC, if one exists (even port address).

extCtlReg

is the External Control Register, which is used to program Channels A and/or B of the MPSC for secondary transmit data or external clocks.

earPort

is the Extended Address Register (EAR) used when mapping CPU memory onto the X-Bus memory map. (Consult the appropriate processor manual for details.)

parityEnablePort

contains a bit to enable/disable a memory parity error NonMaskable Interrupt (NMI) on a workstation. On some processors, this port also contains bits to enable/disable video bit maps and to enable/disable mapping of the CPU boot ROM.

parityErrPort0

returns the low 16 bits of the memory or I/O address that was latched as a result of the last NMI on a workstation.

parityErrPort1

returns the upper bits of the memory or I/O address as a result of an NMI on a workstation. It also contains bits used to determine the cause of the NMI. (Bits in this register differ with different processors. See the processor technical references for details.)

vidControl

is used to enable/disable the video display. On some processors, this is the same port as BeeperPort.

vid6845Addr

is the register used to select data registers that are internal to the 6845 video controller. This value is valid only on those processors that include a 6845 video controller.

vid6845Data

is the register used to write to selected data registers that are internal to the 6845 video controller. This value is valid only on those processors that include a 6845 video controller.

commDmaAddr

is the memory address register in the 8237 DMA controller that is used for intraccluster (RS-422) communications.

commDmaWrdCnt

is the word count register in the 8237 DMA controller that is used by RS-422 cluster communications.

commDmaEar

is the register that contains the upper memory address bits for intraccluster (RS-422) DMA transfers. (See the appropriate processor manual for a description.)

dmaMask

is the channel mask register in the 8237 DMA controller.

dmaMode

is the mode register in the 8237 DMA controller.

dmaCommand

is the command register in the 8237 DMA controller.

dmaBytePtrClr

is the register in the 8237 DMA controller used to clear the first/last byte flip-flop.

rdsrL

is the data port in the 2652 Multi-Protocol Communications Circuit (MPCC) used in intraccluster (RS-422) communications.

rdsrH

is the 2652 controller receive status.

tdsrL

is the 2652 controller transmit data register.

tdsrH

is the 2652 controller transmit command and transmit status register.

pcsarL

is the 2652 controller synchronization/address register.

pcsarH

is the 2652 controller parameter control register.

pcrL

does not exist. This port returns zeros when read.

pcrH

is the 2652 controller character length register.

cPort

is the control register for the RS-422 cluster control state machine. This register controls DMA, interrupts, and the transmit clock.

stat

is the register that returns the status of the RS-422 cluster state machine. (See the processor manuals for a complete description of this register's status bits.)

timerCtl

is the 8254 counter mode register, which generates the RS-422 cluster communications clock.

timerComm

is the 8254 counter load register, which generates the RS-422 cluster communications clock.

clearRtcIntPort

clears a pending Realtime Clock interrupt when read or written (depending upon processor type).

xIntR3vector

is the interrupt vector (not an I/O port) used for level three X-Bus interrupts.

processorType

is not a port address but is a value used to determine the workstation processor type upon which the operating system is running. The values of *processorType* are:

Value	Processor type
4	186
5	286/386/B28/B38
7	CWS
9	286i/386i/B39
14	SuperGen Series 2000
15	SuperGen Series 5000

dmaEarEnable

is a register containing bits to control the DMA EARs in some processors, such as the 286i and 386i. Bits are provided to enable an EAR for a specific channel as well as to specify that channel as either a word or byte channel. (See the specific processor manual for details.)

timeClockBaseAddr

is the first I/O port in a Realtime Clock chip, if the processor includes one. (See the specific processor manual for details.)

nonVolatileRamBaseAddr

is the first I/O port in a non-volatile random access memory block, if the processor includes one. (See the specific processor manual for details.)

protectedModeEnabl

when written to, causes the processor memory system to be configured for operation in protected mode.

fingerPort

when written to, causes an 80286-based processor to be switched from protected mode to real mode.

shCpuSpeed

is not a port address. *shCpuSpeed* (lower byte) contains a value scaled to the instruction execution speed of the processor, and is used in some operating system timing loops.

videoType

is not a port address and is for internal use only. *videoType* is a variable used by the video initialization portion of the operating system.

rs232DmaAddr0

is the address register for channel 0 of the 8237 DMA controller used by the receive portion of RS-232-C channel A.

rs232DmaWrdCnt0

is the word count register for the DMA controller channel 0 (RS-232-C receive channel A).

rs232DmaAddr1

is the address register for channel 1 in the DMA controller used by RS-232-C channel A transmit.

rs232DmaWrdCnt1

is the word count register for the DMA controller channel for RS-232-C channel A transmit.

rs232DmaAddr2

is the address register for channel 2 in the DMA controller used by RS-232-C channel B receive.

rs232DmaWrdCnt2

is the word count register for the DMA controller channel for RS-232-C channel B receive.

rs232DmaAddr3

is the address register for channel 3 in the DMA controller used by RS-232-C channel B transmit.

rs232DmaWrdCnt3

is the word count register for the DMA controller channel for RS-232-C channel B transmit.

rs232DmaCommand

is the DMA controller command register used for all RS-232-C channels in the processor.

rs232DmaRequest

is the DMA controller request register used for RS-232-C.

rs232DmaMask

is the RS-232-C DMA controller individual mask register.

rs232DmaMode

is the RS-232-C DMA controller mode register.

rs232DmaBytePtrClr

is the byte pointer clear register in the 8237 DMA controller used by the RS-232-C channels.

rs232DmaMstrClr

is the master clear register in the RS-232-C DMA controller.

rs232DmaMaskAll

is the all-channel mask register in the RS-232-C DMA controller.

rs232DmaEar0

is the EAR for RS-232-C DMA channel 0.

rs232DmaEar1

is the EAR for RS-232-C DMA channel 1.

rs232DmaEar2

is the EAR for RS-232-C DMA channel 2.

rs232DmaEar3

is the EAR for RS-232-C DMA channel 3.

rs232DmaEarEnable

selects/enables the EARs for RS-232-C DMA.

ledDisplayPort

controls a single 7 segment LED display on some processors.

This page intentionally left blank

Description

The *Process Control Block (PCB)* is the root structure containing the combined hardware and software context of a process executing on a real mode operating system.

The hardware context of a process consists of values to be loaded into processor registers when the process is scheduled for execution. This includes the registers that control the location of the process's stack.

The software context of a process consists of its default response exchange, the priority at which it is scheduled for execution, and the interrupt vectors pointing to software interrupt handlers that the program uses.

When a process preempts another process of lower priority, the operating system saves the hardware context of the preempted process in that process's PCB. When the process is rescheduled for execution, the operating system restores the content of the registers, thus permitting the process to continue as though it were never interrupted.

Related System Structures

- Extended Process Control Block
- Port Structure
- System Configuration Block

This page intentionally left blank.

(continued)

Process Control Block

Offset	Field	Size (bytes)	Description
0	oPcbLnk	2	next process in the run queue
2	status	1	status byte
3	priority	1	process priority
4	ldt	2	real mode SP;protected mode LDT selector
6	tss	2	real mode SS;protected mode TSS selector
8	pmRetAddr	4	address of Wait/Check return pointers
12	defaultRespExch	2	default response exchange
14	userNum	2	user number of process
16	oExPcb	2	offset to Extended Process Control Block

oPcbLnk

is the identifier of the next process in the Run Queue. The *Run Queue* lists the processes that are ready to run in priority order.

status

is the process status byte. The meanings of the *status* bits when set are as follows:

Bit	Meaning when set
0	process is on the run queue
1-4	suspend count. When this count is greater than 0, the process is suspended.
5	process is waiting for a critical section semaphore.
6	process is a system service
7	PCB for process is valid

priority

is the process priority. The priority is a number from 0 to 255, with 0 the highest priority.

ldt

is one of the following values:

Value	Meaning
0	process is GDT-based
1	process is an RMOS process (real mode process running on a protected mode operating system)
any other value	LDT selector of the process

tss

is the TSS of the process.

msgRetAddr

is the memory address at which the Kernel primitives Wait and Check return pointers.

defaultRespExch

is the default response exchange of the process.

userNum

is the user number of the process.

oExPcb

is the offset to the Extended Process Control Block (EPCB). See "Extended Process Control Block," earlier in this chapter.

Description

The *queue entry header* is contained in the first 40 bytes of each queue entry in a queue file. The header is reserved for the Queue Manager and includes control information for linking queue entries together. The queue status block is derived from the queue entry header structure. (For details, see "Queue Status Block," later in this chapter.

Related Queue Manager Structures

- Queue File Header
- Queue Status Block

This page intentionally left blank

Queue Entry Header

Offset	Field	Size (bytes)	Description
0	<i>idxSelf</i>	2	block index of this queue entry
2	<i>uniqId</i>	2	unique ID for this queue entry
4	<i>chainType</i>	1	link to list of queued/available entries
5	<i>priority</i>	1	priority (0 to 9) of this queue entry
6	<i>nxtUniqId</i>	2	unique ID of next queue entry in the queue
8	<i>prevIdx</i>	2	block index of the preceding queue entry
10	<i>nextIdx</i>	2	block index of the next queue entry
12	<i>dateTime</i>	4	indicates when queue entry is to be used
16	<i>repeatInt</i>	2	repeat interval for rescheduling entry
18	<i>markUNum</i>	2	user number of current queue server
20	<i>prevDTIdx</i>	2	block index of the next earlier queue entry
22	<i>nextDTIdx</i>	2	block index of the next later queue entry
24	<i>reserved</i>	16	

idxSelf

is the block index of this queue entry.

uniqId

is a unique identification number for this queue entry. *idxSelf* (above) and *uniqId* are the queue entry handle for this entry.

chainType

indicates whether this entry is linked to the list of queued or available entries.

priority

is the priority (0 to 9 with 0 the highest) of this queue entry.

nxtUniqId

is a unique identification number of the next queue entry in the queue.

prevIdx

is the block index of the preceding queue entry in the queue.

nextIdx

is the block index of the next queue entry in the queue. *nxtUniqId* (above) and *nextIdx* are the queue entry handle of the next entry.

dateTime

is a time stamp used to determine when this queue entry is to be made available for use.

repeatInt

is the repeat interval to be used in rescheduling this queue entry.

markUNum

is the user number of the queue server that currently is using (has marked) this queue entry.

prevDTIdx

is the block index of the queue entry with the next earlier time stamp.

nextDTIdx

is the block index of the queue entry with the next later time stamp.

Description

The *queue file header* is the header portion of a queue. It contains all the information that the Queue Manager needs to manage the queue entries in the queue, such as

- the queue type, such as spooler or RJE
- the queue version
- a listing of all current queue servers
- two sets of head and tail pointers to a doubly linked list of queue entries

As a consistency check, the Queue Manager matches the queue type against the type specified in all client and server operations.

The Queue Manager checks the queue version of existing queues against the Queue Manager version. Note that all queues created by an earlier version of the Queue Manager are interpreted correctly and thus can be managed by a later Queue Manager version. The reverse, however, is not true: queues created by a later Queue Manager cannot be used by an earlier Queue Manager version.

Two sets of head and tail pointers point to a doubly linked list of queue entries. One set contains the addresses of the first and last entries available for use; the other contains the addresses of the first and last entries currently being served or waiting to be served.

Related Queue Manager Structures

Queue Entry Header
Queue Status Block

This page intentionally left blank

(continued)

Queue File Header

Offset	Field	Size (bytes)	Description
0	version	4	current version string for the queue
4	queueType	2	value from 1 to 255
6	fUnique	1	queue access by only one queue server
7	cClients	1	number of queue servers for this queue
8	freeTop	2	first free block to hold a new queue entry
10	queueTop	2	block index of the first queued entry
12	freeBot	2	last free block to hold a new queue entry
14	queueBot	2	block index of the last queued entry
16	priTops	20	linked list of one-word block indexes
36	priNext	20	linked list of one-word block indexes
56	reserved	4	
60	rgUserNum	64	queue server user numbers for this queue
124	rgUserNumQd	64	queue servers waiting for queue entries
188	earliestDT	4	earliest time for delayed/repeating entry
192	earliestId	2	queue entry with the time stamp <i>earliestDT</i> .
196	latestDT	4	latest time for delayed/repeating entry
200	latestId	2	queue entry with the time stamp <i>latestDT</i> .
202	fStable	1	queue file properly closed when last used
203	cQdEntries	2	is the number of queue entries
205	reserved	8	

version

is the current version string for the queue (for example, 11.3).

queueType

is a value from 1 to 255. Types 1, 2, 3, and 4 are assigned as follows:

Type	Assignment
1	spooler queue
2	BSC 2780/3780 RJE queue
3	Batch queue
4	SNA RJE queue

fUnique

is a flag that is TRUE if only one queue server is allowed to access the queue.

cClients

is the number of programs currently established as queue servers for this queue.

freeTop

is the block index of the first free block (available to contain a new queue entry).

queueTop

is the block index of the first queued entry.

freeBot

is the block index of the last free block (available to contain a new queue entry).

queueBot

is the block index of the last queued entry.

priTops

is a linked list of block indexes (one word each) of the first queue entry for a given priority. (The first word contains the first block index with a priority 0 entry, if any. The last word contains the first block index with a priority 9 entry.) Priorities for which there are no entries contain the value 0.

priNext

is a linked list of block indexes (one word each) of the first queue entry with the next highest priority.

rgUserNum

is an array of user numbers of programs currently established as queue servers for this queue.

rgUserNumQd

is an array of user numbers of queue servers waiting for available queue entries.

earliestDT

is the earliest time stamp for any delayed or repeating queue entry in the queue.

earliestId

is the block index of the queue entry with the time stamp, *earliestDT*.

latestDT

denotes the latest time for any delayed or repeating queue entry in the queue.

latestId

is the block index of the queue entry with the time stamp, *latestDT*.

fStable

is a flag that is TRUE if the queue file was properly closed when it was last used.

cQdEntries

is the number of entries in the queue file.

This page intentionally left blank

Description

For each queue entry in a queue (file), there is a *queue status block*. The queue status block is derived from the queue entry header. This structure reports a queue entry's server user number (if the entry is marked), its priority, and the buffers in which the queue entry handle for the queue entry as well as the logically following queue entry are stored.

Related Queue Manager Structures

- Queue Entry Header
- Queue File Header

This page intentionally left blank

(continued)

Queue Status Block

Offset	Field	Size (bytes)	Description
0	qehRet	4	buffer for queue entry handle
4	priority	1	priority of the queue entry
5	serverUserNum	2	user number of the current queue server
7	qehNextRet	4	buffer for next queue entry handle

qehRet

is the buffer in which the queue entry handle of the queue entry is stored.

priority

is the priority (0 to 9, with 0 the highest) at which the queue entry is placed in the queue.

serverUserNum

is the user number of the queue server that currently is using (has marked) this queue entry. This field contains 0FFFFh if the queue entry currently is not marked.

qehNextRet

is the buffer in which the queue entry handle of the logically following queue entry is stored.

This page intentionally left blank

Repeat Attributes Table

Description

The *Repeat Attributes Table* defines keys that repeat when held down. Each entry in the table corresponds to a single key. The first entry corresponds to the key with a keyboard code of 0; the second entry corresponds to the key with a keyboard code of 1, and so forth.

Within each entry, bits 3 through 6 specify the repeat rate of the key, as well as the amount of time before the key begins repeating. Bit 2, when set, indicates that the key is always a command key.

The Repeat Attributes Table resides in a translation data block. The *oAttributes* field of the Translation Data Block Header contains the offset, from the beginning of the data block, of the Repeat Attributes Table. The *cKeys* field of the same header contains the number of entries in the Repeat Attributes Table.

To create or modify a repeat attributes table, use the Keyboard Customization Tool.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Repeat Attributes Table

Offset	Field*	Size (bytes)	Description
0	repeatAttrKey0	2	repeat attribute for keyboard code 0
2	repeatAttrKey1	2	repeat attribute for keyboard code 1
.			
.			
.			

*A repeat attributes table contains a variable number of entries.

repeatAttrKey0

is the repeat attribute for the key with a keyboard code of 0.

repeatAttrKey1

is the repeat attribute for the key with a keyboard code of 1.

.

.

.

and so forth.

This page intentionally left blank

Description

A Resource Descriptor contains information about a resource, such as size, logical file address, type code, and resource ID. The following is the format of a Resource Descriptor.

Resource Descriptor

(continued)

Offset	Field*	Size (bytes)	Description
0	lfaStartResrc	4	Start of file to beginning of resource.
4	cbResource	4	Count of bytes.
8	ResourceId	4	Resource type and resource ID
12	ResourceFlags	4	Reserved
16	qfhSrc	4	File handle of the file the resources are located in.
20	fRsrcData	4	Flag indicating the kind of file handle of the resource.

lfaStartResrc

is the start of the file to beginning of the resource.

cbResource

is the count of bytes on the resource.

ResourceId

is the type and ID of the resource.

ResourceFlags

is reserved.

qfhSrc

is the file handle of the file that the resource is located in.

fRsrcData

is a flag that indicates the kind of file handle of the file that the resource is located in (examples would be a CD-ROM file or a file system file).

Description

The *Special Keys Table* is used by the keyboard process to identify special keys for a given keyboard. This table contains the unencoded value of each special key. These unencoded values are contained in an array of two-byte entries.

A special key is any key that has special significance to an application. Examples of special keys include **Action**, **Lock**, left **Code**, and right **Code**.

The order of the special key entries cannot be changed. For example, the third entry always corresponds to the **Cancel** key; the fifth entry always corresponds to the left **Code** key, and so forth. In a special keys table, between 22 and 29 entries may be present. The first 25 entries represent predefined keys; however, the application can define special keys in the last 6 entries.

The Special Keys Table is a support table that resides in both a Translation Data Block and an Emulation Data Block. The data block offset of the Special Key Table is located in the *oSpecialKeys* field of the Translation or Emulation Data Block Header.

With the Keyboard Customization Tool, the user can construct or modify a Special Keys Table.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic MasksTable
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Repeat Attributes Table
- Table Descriptor Array
- Translation Data Block Header
- Translation Table

(continued)

Special Keys Table

Offset	Field	Size (bytes)	Description
0	aUnencValue	2	unencoded value for a
2	bUnencValue	2	unencoded value for b
4	cancelUnencValue	2	unencoded value for Cancel
6	OvertimeUnencValue	2	unencoded value for Overtime
8	leftCodeUnencVal	2	unencoded value for left Code
10	rightCodeUnencVal	2	unencoded value for right Code
12	finishCodeUnencVal	2	unencoded value for Finish
14	actionCodeUnencVal	2	unencoded value for Action
16	lockCodeUnencVal	2	unencoded value for Lock
18	leftShiftUnencVal	2	unencoded value for left Shift
20	rightShiftUnencVal	2	unencoded value for right Shift
22	reserved	2	reserved for OEMs
24	reserved	2	reserved for OEMs
26	reserved	2	reserved for OEMs
28	cUnencVal	2	unencoded value for c
30	goUnencVal	2	unencoded value for Go
32	nullUnencVal	2	unencoded value for Null
34	diacritEscUnencVal	2	unencoded value for diacritic- Escape
36	sysUnencVal	2	unencoded value for Sys
38	keyboardProcess1	2	reserved for keyboard process
40	keyboardProcess2	2	reserved for keyboard process
42	keyboardProcess3	2	reserved for keyboard process

aUnencVal

is the unencoded value for the **a** special key.

bUnencVal

is the unencoded value for the **b** special key.

cancelUnencVal

is the unencoded value for the **Cancel** special key.

overtimeUnencVal

is the unencoded value for the **Overtime** special key.

leftCodeUnencVal

is the unencoded value for the left **Code** special key.

rightCodeUnencVal

is the unencoded value for the right **Code** special key.

finishCodeUnencVal

is the unencoded value for the **Finish** special key.

actionCodeUnencVal

is the unencoded value for the **Action** special key.

lockCodeUnencVal

is the unencoded value for the **Lock** special key.

leftShiftUnencVal

is the unencoded value for the left **Shift** special key.

rightShiftUnencVal

is the unencoded value for the right **Shift** special key.

reserved

is reserved for OEMs.

reserved

is reserved for OEMs.

reserved

is reserved for OEMs.

cUnencVal

is the unencoded value for the **c** special key.

goUnencVal

is the unencoded value for the **Go** special key.

nullUnencVal

is the unencoded value for the special key defined as **Null**.

diacritEscUnencVal

is the unencoded value defined for the diacritic-**Escape** special key.

sysUnencVal

is the unencoded value for the **Sys** special key.

keyboardProcess1

is reserved for use by the keyboard process.

keyboardProcess2

is reserved for use by the keyboard process.

keyboardProcess3

is reserved for use by the keyboard process.

This page intentionally left blank

Description

The following is the format of a *Spooler control queue entry*. For details on the Spooler service, see the *CTOS Programming Guide*.

Related File Structures

- Spooler Scheduling Queue Entry
- Spooler Status Queue Entry

This page intentionally left blank

(continued)

Spooler Control Queue Entry

Offset	Field	Size (bytes)	Description
0	bCommand	1	command to the spooler
1	restartPage	2	restart printing from this page number
3	sbWpRestartPage	13	description of page from which to restart

bCommand

is the command to the spooler. The command values are

Value	Description
0	Halt/pause printer
1	Cancel print
2	Restart printer
3	Align forms

restartPage

is the page number from which to restart printing. If this value is 0, the printing restarts at the beginning of the current page. If this value is 0FFFFh, the printing starts at the next character in the file.

sbWpRestartPage

is the name describing the page from which to restart printing (for example, a Roman numeral). If the first byte in this entry is 0, this field is ignored.

This page intentionally left blank

Spooler Scheduling Queue Entry

Description

The following is the format of a *Spooler scheduling queue entry*. For details on the Spooler service, see the *CTOS Programming Guide*.

Related File Structures

- Spooler Control Queue Entry
- Spooler Status Queue Entry

This page intentionally left blank

(continued)

Spooler Scheduling Queue Entry

Offset	Field	Size (bytes)	Description
0	fDelAftPrt	1	TRUE deletes spooled file after printing
1	sbFileSpec	92	name of the file to be printed
93	sbFormName	13	name of the form to be used
106	sbWheelName	13	name of the print wheel to be used
119	cCopies	2	number of copies of the file to be printed
121	bPrintMode	1	printing mode
122	fAlignForms	1	TRUE uses the forms alignment option
123	fSecurityMode	1	TRUE prints the file in security mode
124	reserved	5	
129	sbDocName	92	name of the document being printed
221	sbUserName	31	client's user name
252	reserved	4	
256	timeQueued	4	date and time that the print was queued
260	fSupressNewPage	1	TRUE supresses form-feed at start of print
261	fWPPaging	1	TRUE uses WP page escape sequences
262	fSupressBanner	1	TRUE suppresses banner on the notice file
263	fSingleSheet	1	TRUE means printer is manual feed
264	reserved	20	

fDelAftPrt

is a flag. TRUE deletes the spooled file after it is printed.

sbFileSpec

is the name of the file to be printed. The first byte of the sb string is the string length.

sbFormName

is the name of the form to be used. If the length is 0, the standard form will be used.

sbWheelName

is the name of the print wheel to be used. If the length is 0, the standard print wheel will be used.

cCopies

is the number of copies of the file that are to be printed.

bPrintMode

is the printing mode. The values are

Mode	Description
0	Normal
1	Image
2	Binary

fAlignForms

is a flag. TRUE means the forms alignment option will be used.

fSecurityMode

is a flag. TRUE means the file will be printed in security mode.

sbDocName

is the name of the document being printed. This is different from *sbFileSpec*, which is typically a temporary file in the [!Scr]<Spl> directory.

sbUserName

is the client's user name.

timeQueued

are the date and time that the print was queued.

fSupressNewPage

is a flag. TRUE means the Spooler Manager will not print a form-feed at the start of the print.

fWPPaging

is a flag. TRUE means the Spooler Manager will use word processor page escape sequences to determine page numbers.

fSupressBanner

is a flag. TRUE means the Spooler Manager will not print a banner on the notice file.

fSingleSheet

is a flag. TRUE means the printer attached is manual feed.

This page intentionally left blank

Description

The following is the format of a *Spooler status queue entry*. For details on the Spooler service, see the *CTOS Programming Guide*.

Related File Structures

- Spooler Control Queue Entry
- Spooler Scheduling Queue Entry

This page intentionally left blank

(continued)

Spooler Status Queue Entry

Offset	Field	Size (bytes)	Description
0	sbPrinterName	13	name of the printer
13	sbCurrentPage	13	character sequence defining page number
26	reserved	25	
51	sbQueueName	51	name of the queue the printer is serving
102	bChannelNum	1	channel used
103	sbConfigFile	79	printer configuration file name
182	fAtServer	1	TRUE if the system service is at the server
183	bStatus	1	printer status
184	sbSpooledFile	79	name of the currently printing file
263	sbWheelName	13	name of the current print wheel
276	sbFormName	13	name of the current forms
289	sbPauseMessage	61	pause message to be displayed
350	fNeedWheelChange	1	TRUE if a different print wheel is needed
351	fNeedFormsChange	1	TRUE if a different form is needed
352	fShowPauseMsg	1	TRUE means display the pause message
353	wsNum	2	workstation number
355	reserved	2	
357	sbDocName	79	name of the document being printed
436	sbUserName	31	client's name
467	timeStarted	4	date and time that the print was started

sbPrinterName

is the name of the printer.

sbCurrentPage

is the character sequence that defines a page number in a word processor print file.

sbQueueName

is the name of the queue the printer is serving.

bChannelNum

is the channel used. See the *CTOS Operating System Concepts Manual* for a description of the channels.

sbConfigFile

is the name of the printer configuration file.

fAtServer

is a flag. TRUE means the system service is located at the server of a cluster.

bStatus

is the printer status. The status values are

Value	Description
0	Idle
1	Paused
2	Printing
3	Offline
4	Down

sbSpooledFile

is the name of the currently printing file.

sbWheelName

is the name of the current print wheel. If the length is 0, the standard print wheel is being used.

sbFormName

is the name of the current forms. If the length is 0, the standard forms are being used.

sbPauseMessage

is the pause message to be displayed.

fNeedWheelChange

is a flag. TRUE means a different print wheel is needed.

fNeedFormsChange

is a flag. TRUE means a different form is needed.

fShowPauseMsg

is a flag. TRUE means the pause message should be displayed.

wsNum

is the workstation number.

sbDocName

is the name of the document being printed.

sbUserName

is the client's name.

timeStarted

are the date and time that the print was started.

This page intentionally left blank

Description

The *standard file header* is a structure located at the beginning of the first block of a structured file, such as an ISAM, RSAM, or DAM file. The header consists of information common to all structured (standard access method) files followed by information unique to the particular type of file. The first physical record (consisting of the record header, the record data, and the record trailer stored in contiguous bytes) is located at the beginning of the second file block.

The offsets described in the standard file header format are relative to the beginning of the file.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Entry for a Directory in a Master File Directory Block
- Entry for a File in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard Record Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

This page intentionally left blank

(continued)

Standard File Header Format

Offset	Field	Size (bytes)	Description
0	recordHeader	7	standard record header
7	rgbSignature	2	standard file header signature
9	bFileType	1	file type of last access method
10	sPhyRecordMin	2	minimum physical record size
12	sPhyRecordMax	2	maximum physical record size
14	reserved	31	
45	cbAmDependent	2	access-method-dependent information
47	wChecksum	2	checksum of the preceding 40 bytes
49	recordTrailer	1	standard record trailer

recordHeader

is the standard record header with the following contents:

Value	Field
0	qURI
50	sPhyRecord
16	bCheck

For details on each field, see "Standard Record Header Format" in this chapter. The value of *bCheck* changes according to the field description.

rgbSignature

is the standard file header signature, which is the ASCII characters "am".

bFileType

indicates the file type. The type corresponds to the last access method that modified the file. The values of each file type are

Values	File type
2	RSAM
4	DAM
8	ISAM data storage
9	ISAM index

All other values are reserved

sPhyRecordMin

sPhyRecordMax

are the minimum and maximum physical record sizes, respectively, (including the 8-byte standard record header and trailer in the file). DAM can be used to access a file only if *sPhyRecordMin* is equal to *sPhyRecordMax*, which is equal to or greater than 8.

cbAmDependent

is the size of the access-method-dependent information that follows the standard file header.

wChecksum

is a word checksum of the preceding 40 bytes. The standard record header at the beginning of the file header is not included in the checksum.

recordTrailer

is a standard record trailer. See "Standard Record Trailer" in this chapter.

Standard Record Header Format

Description

The *standard record header format* is part of the physical record for a structured file, such as an ISAM, RSAM, or DAM file. The other two parts of the physical record are the record data and the record trailer.

The record header, record data, and record trailer are stored in contiguous bytes starting at the beginning of the second file block. (The first block contains the standard file header.)

The offsets described in the standard record header format are relative to the beginning of a physical record.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Entry for a Directory in a Master File Directory Block
- Entry for a File in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Trailer Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

This page intentionally left blank

Standard Record Header Format

Offset	Field	Size (bytes)	Description
0	qURI	4	Unique Record Identifier
4	sPhyRecord	2	physical record size
6	bCheck	1	record status byte

qURI

is the unique record identifier. This value is the same as the lfa of the physical record.

sPhyRecord

is the size of the physical record, including the record header and trailer.

bCheck

is the status byte that can have one of the following meanings:

Value	Meaning
0	record does not logically exist
1	record previously existed but was deleted
2 to 15	reserved
16 to 255	record logically exists. <i>bCheck</i> is set to 16 the first time it is written and is increased on each subsequent write. <i>bCheck</i> recycles to 16 on overflow.

This page intentionally left blank

Description

The *standard record trailer format* is part of the physical record for a structured file, such as an ISAM, RSAM, or DAM file. The other two parts of the physical record are the record header and the record data.

The record header, record data, and record trailer are stored in contiguous bytes starting at the beginning of the second file block. The first block contains the standard file header.

NOTE: The offset described in the standard record trailer format is relative to the beginning of a physical record. sRecord is the logical record size.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Entry for a Directory in a Master File Directory Block
- Entry for a File in a Directory Block
- Master File Directory Block
- File Header Block
- Log File Record Format
- Standard File Header Format
- Standard Record Header Format
- User Control Block
- Volume Home Block
- Volume Home Block (pre-3.0)

This page intentionally left blank

Standard Record Trailer Format

Offset	Field	Size (bytes)	Description
sRecord+7	bDoubleCheck	1	determines if the record is malformed

bDoubleCheck

is a copy of the *bCheck* field value in the standard record header. (See "Standard Record Header Format.") If *bCheck* and *bDoubleCheck* are not equal, the record is malformed.

This page intentionally left blank

String Masks Table

*NOTE: This table has been renamed in Update Notice 1, August, 1992.
See the Multibyte String Masks Table.*

Deleted with Update Notice 1, August, 1992.

Deleted with Update Notice 1, August, 1992.

This page intentionally left blank.

String Offsets Table

*NOTE: This table has been renamed in Update Notice 1, August, 1992.
See the Multibyte String Offsets Table.*

Deleted with Update Notice 1, August, 1992.

Deleted with Update Notice 1, August, 1992.

This page intentionally left blank.

*NOTE: This table has been renamed in Update Notice 1, August, 1992.
See the Multibyte Strings Table.*

Deleted with Update Notice 1, August, 1992.

Deleted with Update Notice 1, August, 1992.

Deleted with Update Notice 1, August, 1992.

Description

The *System Configuration Block* contains detailed information about the System Image (workstation configuration and system build parameters). The System Configuration Block is located in the system partition. Its address is obtained by calling the GetPStructure operation with a *structCode* value of 2C8h. (For details, see the description of GetPStructure in Chapter 3, "Operations.")

Related System Structures

- Extended Process Control Block
- Port Structure
- System Configuration Block

This page intentionally left blank

(continued)

System Configuration Block

Offset	Field	Size (bytes)	Description
0	systemBuildType	1	type of system build (used internally).
1	osType	1	type of operating system
2	saMinLL	2	address of lower end of LL memory
4	saCurrLL	2	address of higher end of LL memory
6	saCurrSL	2	address of lower end of SL memory
8	saMaxSL	2	address of higher end of SL memory
10	saMemMax	2	first byte above installed system memory
12	cPcb	2	number of Process Control Blocks
14	cExch	2	number of exchanges
16	cLinkBlk	2	number of link blocks
18	reserved	1	
19	fLfsToMastr	1	allows sharing server run files in cluster
20	cTrb	2	number of timer request blocks
22	clob	2	number of I/O Blocks
24	cFcb	2	number of File Control Blocks (FCBs)
26	cVhb	2	number of Volume Home Blocks (VHBs)
28	cUcb	2	number of User Control Blocks (UCBs)
30	cUfb	2	number of User File Blocks (UFBs)
32	hardwareType	1	workstation model/SRP processor board
33	clusterConfig	1	type of configuration:
34	fNoFileSystem	1	local file system or not
35	fCommIop	1	commIOP or not
36	partitionType	1	type of partition

systemBuildType

is the type of system build (used internally).

osType

is the type of operating system. *osType* can have one of the following meanings:

Value	Meaning
0	swapping
1	resident

saMinLL

is the segment base address of the lower end of long-lived memory.

saCurrLL

is the segment base address of the higher end of long-lived memory (lower end of free memory).

saCurrSL

is the segment base address of the lower end of short-lived memory (higher end of free memory).

saMaxSL

is the segment base address of the higher end of short-lived memory.

saMemMax

is the segment base address of first byte above installed system memory.

cPcb

is the number of Process Control Blocks (PCBs).

cExch

is the number of exchanges.

cLinkBlk

is the number of link blocks.

fLfsToMaster

is a flag that indicates whether the system configuration file option *LfsToMaster* is being used. This option allows server files to be shared by workstations in the cluster. For details on this option, see "File Management" in the *CTOS Operating System Concepts Manual*. Configuring options is described in the *CTOS System Administration Guide*.

cTrb

is the number of timer request blocks (TRBs).

cIob

is the number of I/O Blocks (IOBs).

cFcb

is the number of File Control Blocks (FCBs).

cVhb

is the number of Volume Home Blocks (VHBs).

cUcb

is the number of User Control Blocks (UCBs).

cUfb

is the number of User File Blocks (UFBs).

hardwareType

is the hardware type. The values of *hardwareType* are

Workstation model

Value	Type
0	IWS
1	AWS-210
2	AWS-220 and -230
3	AWS-240
4	NGEN or B Series workstation*

Shared resource processor board

Value	Type
10	File Processor (FP)
11	Terminal Processor (TP)
12	Cluster Processor (CP)
13	Storage Processor (SP)
14	Data Processor (DP)
20	General Processor (GP)
21	General Processor with SCSI interface (GP+SI)
22	General Processor with communications interface (GP+ CI)

(continued)

System Configuration Block

Value	Type
64	Obsolete (Applications Processor)
65	Obsolete (Applications Processor2)

For the NGEN or B Series types of workstation processor, see the *processorType* field in "Port Structure" in this chapter.

clusterConfig

is the type of configuration. The values of *clusterConfig* are

Value	Description
0	standalone
1	cluster
2	server

fNoFileSystem

is a flag that is TRUE if there is a local file system; FALSE, if no local file system.

fCommIop

is a flag that is TRUE if with CommIOP; FALSE, if not.

bPartitionType

is the type of memory management for partitions. The values of *bPartitionType* are

Value	Description
0FF	Multipartition
3	Variable partition

The field *bPartitionType* contained an additional value (value 1) for the single partition operating system version. The single partition system predated the multipartition and virtual partition systems described in "Partitions and Partition Management" in the *CTOS Operating System Concepts Manual*.

Description

The *System Date/Time Structure* represents the system date and time to greater precision than 1 second. If a program executing on a server or standalone workstation needs to know the time to this precision, it can access the structure by calling the `GetPStructure` operation with a *structCode* value of 240.

The fields *seconds* and *dayTimes2* (System Date/Time format) are returned by the `GetDateTime` operation and can be changed using `SetDateTime`. (For details on all the formats in which the system date and time can be represented, see "Utility Operations" in the *CTOS Operating System Concepts Manual*.)

Related Timer and Interrupt Handler Structures

- Expanded Date/Time Format
- Low Memory Allocation
- Timer Pseudointerrupt Block
- Timer Request Block Format

This page intentionally left blank

(continued)

System Date/Time Structure

Offset	Field	Size (bytes)	Description
0	ticks	1	depends on timer circuitry. resolution
1	hundredMsec	1	count of 100 ms since last second
2	systemDateTimeFormat	4	see below

ticks

is a value that depends on the resolution of the processor timer circuitry.

hundredMsec

is the count (0–9) of 100 ms since the last second.

System Date/Time Format

The system date/time format consists of the following fields:

seconds

is a word value that specifies the count (0–43199) of seconds since last midnight/noon.

dayTimes2

is a word value. The high-order 15 bits contain the count (a value in the range of 0 through 65535) of 12-hour periods since March 1, 1952. This allows dates up to the year 2042 to be represented (0 = null date/time). The low-order bit of this word represents AM (value 0) or PM (value 1).

This page intentionally left blank

Description

The *Table Descriptor Array* within a translation data block is used by the keyboard process to identify the translation data block offset and size of each translation table. In an emulation data block, the Table Descriptor Array only identifies the offset of each emulation table.

The Translation and Emulation Data Blocks reside both in memory and in the file NlsKbd.sys. To obtain the data block offset of the Table Descriptor Array, an application can read either the *pRgTransTableDescs* field in the Translation Data Block Header or the *pRgEmulTableDescs* field in the Emulation Data Block Header.

To determine the number of entries in the Table Descriptor Array, a program can read either the *cTransTablesDescs* in the Translation Data Block Header or the *cEmulTablesDescs* field in the Emulation Data Block Header.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Translation Data Block Header
- Translation Table

(continued)

Table Descriptor Array

Offset	Field*	Size (bytes)	Description
varies	descTable1	varies	offset of 1st table
varies	descTable2	varies	offset of 2nd table
.			
.			
.			

descTable1

is the first entry. In a translation data block, this entry is a double word that contains the data block offset (upper word) and size (lower word), in bytes, of the 1st translation table. In an emulation data block, this entry is a word that contains the data block offset of the 1st emulation table.

descTable2

is the second entry. In a translation data block, this entry is a double word that contains the data block offset (upper word) and size (lower word), in bytes, of the 2nd translation table. In an emulation data block, this entry is a word that contains the data block offset of the 2nd emulation table.

.

.

.

and so forth.

This page intentionally left blank

Telephone Service Configuration Block

Description

The *Telephone Service Configuration Block (TSCB)* provides default values to the Telephone Service during initialization. This structure is contained in the Telephone Service configuration file, which is shared with the Operator program.

In the TSCB, the two byte fields are arrays, one byte for each telephone line. The default values given are used when no configuration file is present.

The *TsSetConfigParams* operation is used to set the configuration parameters in the TSCB. The current configuration information can be queried with *TsQueryConfigParams*.

See Chapter 3, "Operations," for a description of the Voice/Data services operations. For details on how to write applications to be used with Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice Control Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued)

Telephone Service Configuration Block

Offset	Field	Size (bytes)	Description
0	rgDtmfGenOff	2	time between DTMF tones
2	rgDtmfGenOn	2	time during DTMF tones
4	rgFlashTime	2	length of a default flash ("@" in dial string)
6	rgPauseTime	2	length of a default pause ("~" in dial string)
8	rgfPulseDial	2	TRUE means pulse dialing is used
10	rgRingHz	4	workstation monitor ring signal frequency
14	rgiRingThrough	2	ring number in which the ring current sent
16	rgRingMode	2	ring mode for each line
18	rgCodecMode	2	CODEC mode
20	nSecHoldRing	2	ring if on hold this many seconds
22	nSecHoldHangup	2	hang up if on hold this many seconds
24	rgbConfigfile	100	Telephone Service configuration file name
124	rgAction	64	action key values mapped to TsDoFunction
188	rgFunction	64	functions executed when action key typed

rgDtmfGenOff
rgDtmfGenOn

specify the time between (*rgDtmfGenOff*) and during (*rgDtmfGenOn*) DTMF tones when dialing, in units of 10ms. Most PBX's can handle values of 10 (100ms), and some as little as 6 (60ms). The default is 6 for *rgDtmfGenOff* and 8 for *rgDtmfGenOn*.

rgFlashTime

specifies the length of a default flash ("@" character in a dial string) in units of 100ms. Most PBX's use a value of 10 (1 second). The default is 10.

rgPauseTime

specifies the length of a default pause ("~" character in a dial string) in units of 100ms. The default is 20.

rgfPulseDial

is a flag. If TRUE, pulse dialing is used instead of DTMF generation. The default is FALSE.

rgRingHz

is an array used to specify the frequency of the workstation monitor's ringing signal when the telephone line is ringing. A value of 0 means no tone is generated. The range of frequencies is 1 to 255.

Byte	Description
0	neither line (default 0)
1	line 1 (default 40)
2	line 2 (default 90)
3	both lines 1 and 2 (default 150)

rgiRingThrough

is the ring number in which the ring current will be sent directly to the telephone unit. A value of 0FFFFh means the ring current is never passed through. The default is 4.

rgRingMode

is the ring mode for each line. When a line rings, some combination of monitor ringing and telephone unit ringing is done. The ring mode values are

Value	Description
0	Do not ring either the telephone unit or the monitor.
1	Ring only the telephone unit.
2	Ring only the monitor.
3	Always ring both the telephone unit and the monitor.

(continued)

Telephone Service Configuration Block

Value	Description
4	Ring the monitor for the number of rings specified in <i>rgiRingThrough</i> above, then ring the telephone unit.
5	Ring the monitor for the number of rings specified in <i>rgiRingThrough</i> above, then ring both the monitor and the telephone unit (default).

rgCodecMode

is the CODEC mode (always 0).

nSecHoldRing

ring if on hold this many seconds (default is 300).

nSecHoldHangup

hang up if on hold this many seconds (default is 600).

rgbConfigfile

is the Telephone Service configuration file name.

rgAction

is an array of action key values (encoded keyboard values) to be mapped to TsDoFunction functions. A value of 0 terminates the list.

rgFunction

is an array of functions (see TsDoFunction) to be executed when the corresponding action key listed in *rgAction* is typed.

This page intentionally left blank

Telephone Service Configuration File Format

Description

The *Telephone Service Configuration File format* is shown next. The configuration file is shared with the Operator program. The first 256 bytes are used by the Telephone Service and the remainder of the file is used by the Operator.

If no configuration file is present, the Telephone Service is initially configured so that ACTION-0, 1, 2, and . correspond to the TsDoFunction operation values 5, 2, 3 and 14, respectively. Changing the value of the field *rgAction* and/or the field *rgFunction* in the Telephone Service Configuration Block overrides the default configuration.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration Block
- Telephone Status Structure
- Voice Control Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued) **Telephone Service Configuration File Format**

Offset	Field	Size (bytes)	Description
0	signature	2	always 'tC'
2	version	2	configuration file version number
4	TsConfig	252	configuration information
256	OperatorConfig	768	used for storing operator configuration

signature

is always 'tC'.

version

is the version number of the configuration file (currently 1).

TsConfig

is configuration information. For the format of this information, see "Telephone Service Configuration Block," in this chapter.

operatorConfig

is an area used by the Operator program to store configuration information such as the phone number of each line, dialing prefixes, and so forth.

This page intentionally left blank

Description

The *Telephone Status Structure* describes the state of the Voice Processor hardware. The information can be obtained by calling the `TsGetStatus` operation. `TsGetStatus` can be used to obtain the status immediately or the next time a Telephone Service event occurs.

For a description of the Voice/Data Services operations, see Chapter 3, "Operations." Details and examples of how to use these operations are contained in the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Voice Control Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued)

Telephone Status Structure

Offset	Field	Size (bytes)	Description
0	iEvent	2	incremented with status structure change.
2	defaultLine	1	default line as selected by TsDoFunction.
3	fNeedCodecConnection	1	TRUE means a voice operation is waiting
4	fCodecInUse	1	CODEC is recording or playing back
5	pVPCB	3	24-bit physical address of the VPCB
8	reserved	2	
10	baudRate	2	either 300 or 1200
12	originateMode	1	either TRUE or FALSE
13	parityMode	1	is 0, 1, 2, 3, or 4
14	lineControlMode	1	either 0 or 1
15	reserved	1	
16	tUnitState	16	telephone unit state (see tUnitState below)
32	rgLineState(2)	32	line 1/line 2 states (see lineState below)
64	codecState	16	CODEC state (see codecState below)

tUnitState Fields

Offset	Field	Size (bytes)	Description
0	fOffHook	1	if TRUE, the telephone unit is offhook
1	rgfTLine(2)	2	line 1/line 2 connection to telephone unit
3	rgfRingThrough(2)	2	line 1/line 2 ring voltage to telephone unit
5	fCodec	1	telephone unit connection to the CODEC
6	fDtmfRec	1	telephone unit DTMF decoder connection
7	fMonitor	1	telephone unit monitor mode connection
8	handle	2	telephone unit connection handle
10	hookThroughMode	1	mode for passing through on/offhook
11	reserved	5	

Telephone Status Structure

(continued)

lineState Fields

Offset	Field	Size (bytes)	Description
0	status	1	line status
1	fDialing	1	dial string is being processed on this line
2	dialState	1	the current state of dialing
3	fRinging	1	if TRUE, the line is ringing
4	iRing	1	number of rings
5	fRingThrough	1	passing ring through to the telephone unit
6	fOffHook	1	if TRUE, the line is offhook
7	fHold	1	if TRUE, the line is on hold
8	fCodec	1	if TRUE, the CODEC is being used
9	fDtmfRec	1	if TRUE, the DTMF detector is being used
10	handle	2	telephone unit connection handle
12	reserved	1	
13	fModem	1	if TRUE, the modem is being used
14	reserved	2	

codecState Fields

Offset	Field	Size (bytes)	Description
0	lfaCurrent	4	lfa of data record last processed
4	qSampleCurrent	4	data byte number last processed
8	reserved	8	

iEvent

is a counter which is incremented every time the contents of the status structure changes.

defaultLine

is the default line as selected by TsDoFunction.

fNeedCodecConnection

is a flag that is TRUE if a voice operation is waiting for a connection.

fCodecInUse

is a flag that is TRUE if the CODEC is recording or playing back voice.

pVPCB

is the 24-bit physical address of the Voice Processor Control Block (VPCB).

baudRate

is either 300 or 1200.

originateMode

is TRUE if the originator or FALSE if the answerer.

parityMode

is the parity control, where the values are

Value	Description
0	none
1	even (bit 7 is set or cleared so that there are always an even number of bits set in each byte)
2	odd (bit 7 is set or cleared so that there are always an odd number of bits set in each byte)
3	1 (bit 7 is always set); also known as 'mark'
4	0 (bit 7 is always cleared); also known as 'space'

lineControlMode

is one of the following values:

Value	Description
0	no flow control
1	XON/XOFF flow control

tUnitState

is the telephone unit state. (See "**tUnitState Fields**," below.)

rgLineState

are the line 1 and line 2 states. (See "**lineState Fields**," below.)

codecState

is the CODEC state. (See "**codecState Fields**," below.)

tUnitState Fields

fOffHook

is a flag that is TRUE if the telephone unit is offhook.

rgfTLine

is an array of flags which, if TRUE mean line 1 and/or line 2 is connected to the telephone unit.

rgfRingThrough

is an array of flags which, if TRUE mean line 1 or line 2 is connected directly to the telephone unit to allow ring voltage to be passed through.

fCodec

is a flag that is TRUE if the telephone unit is connected to the CODEC.

fDtmfRec

is a flag that is TRUE if the telephone unit is connected to the DTMF decoder.

fMonitor

is a flag that is TRUE if the telephone unit is connected in monitor mode.

handle

is the handle associated with the current telephone unit connection.

hookThroughMode

is the mode for passing through on/off hook from the telephone unit to the telephone lines. The values are

Value	Description
0	neither line
1	line 1
2	line 2
3	both lines

lineState Fields

status

is the line status. The values are

Value	Description
0	onhook
1	ring current detected
2	modem not ready
3	modem ready (FSK, 300 baud)
4	modem ready (PSK, 1200 baud)
5	offhook

fDialing

is a flag that is TRUE if a dial string is being processed on this line.

dialState

is the current state of dialing. The values are

Value	Description
0	idle
1	generating DTMF
2	analyzing CPTR (waiting for dial tone)
3	generating pulse
4	generating flash
5	generating pause
6	analyzing CPTR (waiting for any tone)
7	analyzing CPTR (waiting for answer)
255	performing error recovery

fRinging

is a flag that is TRUE if the line is ringing.

iRing

is the number of rings.

fRingThrough

is a flag that is TRUE if the ringing is being passed through to the telephone unit.

fOffHook

is a flag that is TRUE if the line is offhook.

fHold

is a flag that is TRUE if the line is on hold.

fCodec

is a flag that is TRUE if the CODEC is being used.

fDtmfRec

is a flag that is TRUE if the DTMF detector is being used.

handle

is the connection handle associated with the current telephone unit connection.

fModem

is a flag that is TRUE if the modem is being used.

codecState Fields

lfaCurrent

is the logical file address (lfa) of the data record last processed.

qSampleCurrent

is the data byte number last processed.

Description

The *Terminal Output Buffer* defines the area that the client and the operating system use to output data through the port on a TP or CP processor board. (Note that *InitCommLine* described in Chaptr 3, "Operations," is used to perform this function on all other processor boards.) The OS process that manages the port reads the bytes placed in the buffer by the client and outputs them through the port.

For a discussion of the shared resource processor terminal management operations, see "SRP Terminal Management" in the *CTOS Operating System Concepts Manual*.

Related Shared Resource Processor Structures

CPU Description Table

CPU Description Table (pre-3.0)

This page intentionally left blank

(continued)

Terminal Output Buffer

Offset	Field	Size (bytes)	Description
0	ioOutTop	2	header offset to first data byte in the fifo
2	ioOutBot	2	header offset to the bottom of the fifo
4	ioOutGet	2	header offset to the byte to fetch next
6	ioOutPut	2	header offset to the next client byte
8	foLock	1	protects Put pointer from multiple writers
9	foCarrier	1	describes the sense of the line's carrier
10-17	rgbAP	8	reserved for the use of the port clients

ioOutTop

is the offset from header to first data byte in the buffer. The first byte is the one at the lowest address (nearest 0000).

ioOutBot

is the offset from the header to the bottom of the buffer. The bottom byte is one past the data byte occupying the highest address.

ioOutGet

is the offset from the header to the byte the output interrupt routine will fetch next. After this byte is fetched, this pointer is incremented. Thus, the discipline is post-increment.

ioOutPut

is the offset from the header to the next client byte. Like the Get pointer, this pointer is postincremented. Note that if the two pointers are equal, the buffer is empty. If the modulo increment of the Put pointer equals the Get pointer, the buffer is full. All manipulations of the Put pointer must be done with the lock byte set (see below).

foLock

is a semaphore used to protect the Put pointer from multiple writers (including the Terminal Processor itself). Setting the sign of the byte nonzero will keep others from using the Put pointer.

foCarrier

is a Boolean flag describing the sense of the line's carrier. For dataset lines, this flag tracks the DCD (data carrier detect) signal. For hardwired lines, *foCarrier* is always TRUE.

rgbAP

is an 8-byte area reserved for the use of the clients of the port. CTIX uses this field.

Description

A *timer pseudointerrupt block* is a structure allocated in application memory. It is used when the caller wants to establish a pseudointerrupt handler with the SetTimerInt operation. For details, see "Timer Management" in the *CTOS Operating System Concepts Manual*.

Related System Structures

- Expanded Date/Time Format
- Low Memory Allocation
- System Date/Time Structure
- Timer Request Block Format

This page intentionally left blank

(continued)

Timer Pseudointerrupt Block

Offset	Field	Size (bytes)	Description
0	linkField1	4	used by the operating system
4	linkField2	4	used by the operating system
8	pIntHandler	4	handler entry point or an exchange
12	saData	2	DS address to be used by handler
14	cIntervals	2	interval before pseudointerrupt occurs
16	pRqBlkRet	4	address at which TPIB pointer is returned
20	footPrint	2	used by the operating system
22	delta	2	used by the operating system
24	reserved	8	used by the operating system

linkField1

is used internally by the operating system.

linkField2

is used internally by the operating system.

pIntHandler

is either the CS:IP of the entry point of the pseudointerrupt handler, or the exchange where the address of the TPIB (message) is sent when the timer interval expires.

saData

is the segment base address of the data segment to be used by the pseudointerrupt handler.

cIntervals

is the interval (in 50 microsecond units) before the pseudointerrupt is to occur.

pRqBlkRet

is the memory address to which the address of the TPIB is returned when the pseudo-interrupt handler is invoked.

footPrint

is used internally by the operating system.

delta

is used internally by the operating system.

Description

A *timer request block (TRB)* is a block of memory shared between a client and timer management's Realtime Clock (RTC) service. It contains fields for keeping track of time intervals specified by the client. The client initiates a timing session by calling the `OpenRtClock` operation. Every 100 milliseconds, the RTC service examines each open TRB and, if necessary, adjusts the values of fields it contains. When a specified time interval has elapsed, the client is notified by a message that is sent to the response exchange in the TRB.

The TRB format allows a client to time a single interval or to use the RTC service for repetitive timing, in which the client is notified each time a specified interval has elapsed. For details on each of these methods of timing and ways to prevent race conditions when using the TRB for repetitive timing, see "Timer Management" in the *CTOS Operating System Concepts Manual*.

Related Timer and Interrupt Handler Structures

- Expanded Date/Time Format
- Low Memory Allocation
- System Date/Time Structure
- Timer Pseudointerrupt Block

This page intentionally left blank

(continued)

Timer Request Block Format

Offset	Field	Size (bytes)	Description
0	counter	2	decremented every 100 milliseconds
2	counterReload	2	copied to counter field when ctr reaches 0
4	cEvents	2	incremented when counter field reaches 0
6	exchResp	2	response exchange
8	ercRet	2	0 unless a signature (see below) is used
10	rqCode	2	request code supplied by the client

counter

is a value that initially specifies the number of 100-millisecond intervals that are to elapse before the client is notified. This field is decremented every 100 milliseconds.

counterReload

is copied to the *counter* field when the counter reaches 0. A nonzero value specified in this field allows the TRB to be used for repetitive timing.

cEvents

is incremented when the *counter* field is decremented to 0. *cEvents* provides a continuous count of the events that have occurred but are not yet processed by the client. Ways a client can process the correct number of events are described in "Timer Management" in the *CTOS Operating System Concepts Manual*.

exchResp

is the response exchange to which the memory address of the TRB is sent.

ercRet

should be initialized to zeros. The client can optionally assign the 'DZ' signature value to this field. 'DZ' allows the timer to continue ticking when the client who called `OpenRtClock` is swapped out of memory. The Doze operation, for example, places this signature value in the TRB so that a swappable program can maintain accurate timing.

rqCode

is the request code. *rqCode* is not used by timer management, but the client should place a unique value in this field so it can identify its own TRB when the TRB is received as a message.

Description

The *Translation Data Block Header* is located at the beginning of a translation data block. Information in the Translation Data Block Header includes the memory offsets of the support tables within the data block, the number of entries within these tables, and the ID of the attached keyboard.

Examples of translation data block tables include the Allowed States Table and the Command Masks Table. Each translation data block table is also described in this chapter.

The *Translation Data Block Header* provides the keyboard process with the information it needs to access the support tables within the Translation Data Block.

The address of the *Translation Data Block Header* is located in the Partition Keyboard Information Structure, which is also used by the keyboard process.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Table

(continued)

Translation Data Block Header

Offset	Field	Size (bytes)	Description
0	transDataBlock-Signature	2	signature of Translation Data Block
2	transDataBlockID	2	ID of Translation Data Block
4	cbTransHeader	2	size of Translation Data Block Header
6	transDataBlock-Version	2	version number of Translation Data Block
8	cSpecialKeys	2	number of keys in Special Keys Table
10	oSpecialKeys	2	offset of Special Keys Table
12	cChords	2	count of chords in Chords Table
14	oChords	2	offset of Chords Table
16	cControlChords	2	count of entries in Control Chords Table
18	oControlChords	2	offset of Control Chords Table
20	cConditions	2	count of entries in Conditions Table
22	oConditions	2	offset of Conditions Table
24	cKeys	2	count of translation keys
26	oAttributes	2	offset of Repeat Attributes Table
28	oAllowedStates	2	offset of Allowed States Table
30	oCmdKeyMasks	2	offset of Command Masks Table
32	oDiacritMasks	2	offset of Diacritic Masks Table
34	cDiacritics	2	count of diacritics in Diacritics Table
36	oDiacritics	2	offset of Diacritics Table
38	oStringMasks	2	offset of Multibyte String Masks Table
40	cStringOffsets	2	count of entries in Multibyte String Offsets Table
42	oStringOffsets	2	offset of Multibyte String Offsets Table
44	cTransTablesDescs	2	count of entries in Table Descriptor Array
46	oRgTransTable-Descs	2	offset of Table Descriptor Array
48	oDecodingOffset	2	offset of Decoding Offsets Table
50	cbChecked	2	number of bytes included in checksum
52	checksum	4	checksum of data in Translation Data Block

transDataBlockSignature

is 'TH', the signature of the Translation Data Block.

transDataBlockID

is the ID of the Translation Data Block. The high-order byte is always 70h, and the low-order byte is the ID of the corresponding keyboard.

cbTransHeader

is the size of the Translation Data Block Header.

transDataBlockVersion

is the version number of the Translation Data Block.

cSpecialkeys

is the number of keys in the Special Keys Table.

oSpecialKeys

is the offset, from the beginning of the Translation Data Block Header, of the Special Keys Table. (See the Special Keys Table.)

cChords

is the count of chord keys in the Chords Table.

oChords

is the offset, from the beginning of the Translation Data Block Header, of the Chords Table. (See the Chords Table.)

cControlChords

is the count of control chord entries in the Control Chords Table.

oControlChords

is the offset, from the beginning of the Translation Data Block Header, of the Control Chords Table. (See the Control Chords Table.)

cConditions

is the count of condition entries in the Conditions Table.

oConditions

is the offset, from the beginning of the Translation Data Block Header, of the Conditions Table. (See the Conditions Table.)

cKeys

is the number of entries in the Allowed States Table, the Repeat Attributes Table, the Command Masks Table, the Diacritic Masks Table, the Multibyte String Masks Table, and the Translation Tables.

oAttributes

is the offset, from the beginning of the Translation Data Block Header, of the Repeat Attributes Table. (See the Repeat Attributes Table.)

oAllowedStates

is the offset, from the beginning of the Translation Data Block Header, of the Allowed States Table. (See the Allowed States Table.)

oCmdKeyMasks

is the offset, from the start of the Translation Data Block Header, of the Command Masks Table. (See the Command Masks Table.)

oDiacritMasks

is the offset, from the start of the Translation Data Block Header, of the Diacritic Masks Table. (See the Diacritic Masks Table.)

cDiacritics

is the count of diacritic key entries in the Diacritics Table.

oDiacritics

is the offset, from the beginning of the Translation Data Block Header, of the Diacritics Table. (See the Diacritics Table.)

oStringMasks

is the offset, from the beginning of the Translation Data Block Header, of the Multibyte String Masks Table. (See the Multibyte String Masks Table.)

cStringOffsets

is the count of entries in the Multibyte String Offsets Table.

oStringOffsets

is the offset, from the beginning of the Translation Data Block Header, of the Multibyte String Offsets Table. (See the Multibyte String Offsets Table.)

cTransTablesDescs

is the number of translation tables described in the Table Descriptor Array. This array contains the data block offset and size (in bytes) of each translation table in the Translation Data Block.

pRgTransTableDescs

is the offset, from the start of the Translation Data Block Header, of the Table Descriptor Array. (See the Table Descriptor Array.)

oDecodingOffsets

is the offset, from the start of the Translation Data Block Header, of the Decoding Offsets Table. (See the Decoding Offsets Table.)

(continued)

Translation Data Block Header

cbChecked

is the number of bytes, from the beginning of the Translation Data Block Header, that were included in the checksum.

checksum

is an internal value used to verify the integrity of the Translation Data Block.

This page intentionally left blank.

Description

A translation table defines the character code that corresponds to an unencoded value. The first entry contains the corresponding character code for an unencoded value of 0; the second entry contains the corresponding character code for an unencoded value of 1, and so forth. Each table may have entries that are 1, 2, or 4 bytes in length, but all entries in a given table must be the same size. Note that some entries in a translation table may contain indexes to strings in a strings table.

A maximum of 16 translation tables may exist in a translation data block. The data block offset of a translation table is located in the Table Descriptor Array.

For more information on translation tables, see "Keyboard and I-Bus Management" in the *CTOS Operating System Concepts Manual*.

Related Keyboard Structures

- Allowed States Table
- Chords Table
- Command Masks Table
- Conditions Table
- Control Chords Table
- Decoding Offsets Table
- Decoding Table
- Diacritics Table
- Diacritic Masks Table
- Emulation Table
- Emulation Data Block Header
- Emulation LEDs Table
- Repeat Attributes Table
- Multibyte String Masks Table
- Multibyte String Offsets Table
- Multibyte Strings Table
- Special Keys Table
- Table Descriptor Array
- Translation Data Block Header

This page intentionally left blank.

(continued)

Translation Table

Offset	Field*	Size (bytes)	Description
0	charCode0	varies	character code for unencoded value of 0
varies	charCode1	varies	character code for unencoded value of 1
.			
.			
varies	charCode255	varies	character code for unencoded value of 255

*A translation table presently contains 256 entries.

charCode0

is the character code for a key with an unencoded value of 0.

charCode1

is the character code for a key with an unencoded value of 1.

.

.

.

charCode255

is the character code for a key with an unencoded value of 255.

This page intentionally left blank.

Description

The *User Control Block (UCB)* is a file management system data structure that is assigned to each user number. The UCB contains the default node, default volume, default directory, and default password set by the last SetPath operation, and the default file prefix set by the last SetPrefix operation.

A program can obtain a copy of the UCB for parsing or building file specifications by calling the GetUcb operation. (For details, see "File Management" in the *CTOS Operating System Concepts Manual*.)

Related File Structures

None

This page intentionally left blank

(continued)

User Control Block

Offset	Field	Size (bytes)	Description
0	reserved	2	
2	sbDefaultVol	13	default volume
15	sbDefaultDir	13	default directory
28	sbDefaultPassword	13	default password
41	sbPrefix	41	default prefix
82	sbDefaultNode	13	default node

sbDefaultVol

is the default volume. The size of the volume name string is contained in the first byte.

sbDefaultDir

is the default directory. The size of the directory name string is contained in the first byte.

sbDefaultPassword

is the default password. The size of the password string is contained in the first byte.

sbPrefix

is the default prefix. The size of the prefix string is contained in the first byte.

sbDefaultNode

is the default node. The size of the node name string is contained in the first byte.

This page intentionally left blank

Variable Length Parameter Block

Description

The *Variable Length Parameter Block (VLPB)* is a structure created in the long-lived memory of an application partition that stores long-lived data for use by other programs executing in the partition. The purpose of the VLPB is to organize the data in such a way that the data can be easily retrieved.

A VLPB can be created and its contents queried using CTOS operations. For details, see "Parameter Management" in the *CTOS Operating System Concepts Manual*.

The memory address of the VLPB is stored in the Application System Control Block (ASCB) of the partition. To obtain the address of the ASCB, your program can call either GetPAScb or GetPStructure.

Related Partition Structures

- Application System Control Block
- Batch Control Block
- Extended Partition Descriptor
- Partition Configuration Block
- Partition Descriptor

This page intentionally left blank

(continued)

Variable Length Parameter Block

Offset	Field	Size (bytes)	Description
0	sVarParams	2	bytes of LL memory allocated for the VLPB
2	ibFirstFree	2	first free byte in the VLPB
4	CParams	2	count of rows for parameter values
6	rgSdoParam (CParams + 1)	4*(CParams + 1)	array of VLPB row descriptors

sVarParams

is the total number of bytes allocated from long-lived memory for the VLPB.

ibFirstFree

is the first free byte. *ibFirstFree* indicates the number of bytes already used.

CParams

is the count of rows (iparams) in the VLPB array dedicated to storing parameter values. *CParams*, for example, is the number of parameter fields displayed in an Executive command form not including the command name field. (For details, see "Parameter Management" in the *CTOS Operating System Concepts Manual*.)

rgSdoParam

is an array of row descriptors, each of which consists of a pair of words (ob,cb) with the following meanings:

Word	Meaning
ob	is the offset within the VLPB of the corresponding row
cb	is the number of bytes occupied by the row

The size in bytes of this array is $4 * (CParams + 1)$. The extra byte represents the row containing the command name.

Description

The *Video Control Block (VCB)* contains all information known to the operating system about the video display, including the location, height, and width of each frame, and the coordinates at which the next character is to be stored in the frame by Sequential Access Method.

Your program can obtain the address of the VCB by calling the `GetPStructure` operation with a *structCode* value of 2.

Related Video Structures

Frame Descriptor

This page intentionally left blank.

(continued)

Video Control Block

Offset	Field	Size (bytes)	Description
0	level	1	the level of video capability
1	fCharAttrs	1	character attributes enabled/disabled
2	fReverseVideo	1	reverse video enabled/disabled
3	fHalfBright	1	half-bright enabled/disabled
4	pMap	4	address of character map
8	sMap	2	character map size
10	cFrames	1	number of frames set at system build
11	cColsMax	1	screen width
12	cLinesMax	1	screen height
13	sLine	1	bytes required for one character map line
14	defaultWindowId	1	default window identification number
15	reserved	1	
16	bSpace	1	character code of empty space on screen
17	SAR	2	contents of Screen Attribute Register
19	nPixelsWide	2	width in pixels of the bit map
21	nPixelsHigh	2	height in pixels wide of the bit map
23	wsLine	2	same as iLine but is a value for all systems
25	reserved	2	
27	iFrameCursor	1	reserved for future use
28	cLinesMaxScreen	1	maximum lines displayed in this mode
29	fHardwareCharMap	1	TRUE if there is a hardware character map
30	nPlanes	1	number of planes used for color graphics
31	fBackgroundColor	1	sets the background color mode
32	rgbRgFrame	*	array of frame descriptors

*20 bytes for each frame defined at system build.

level

is the level of video capability. *level* has the following values:

Values	Meaning
0	reserved
1	reserved
2	reserved
3	reserved
4	character-map workstations
6	bit-map workstations

Other values will identify future capability levels. This field is set by the QueryVidHdw and ResetVideo operations.

fCharAttrs

is TRUE if the character map includes character attributes and the use of character attributes is enabled in the Screen Attribute Register in the video hardware. It is FALSE otherwise. The *fAttr* parameter to the ResetVideo operation is placed here. It is set by the SetScreenVidAttrs operation.

fReverseVideo

is TRUE if the screen reverse video is enabled in the Screen Attribute Register in the video hardware. This causes the hardware to display dark characters on a light background. It is FALSE otherwise. It is initialized by the ResetVideo operation to FALSE and can be changed by the SetScreenVidAttr operation.

fHalfBright

is TRUE if screen half-bright is enabled in the Screen Attribute Register in the video hardware. This causes the hardware to display at half-bright. It is FALSE otherwise. It is initialized by the ResetVideo operation to FALSE and can be changed by the SetScreenVidAttr operation.

*pMap**sMap*

are the memory address and size of the character map.

cFrames

is the number of frames. This number is established at system build. The default is 8.

*cColsMax**cLinesMax*

are the height and width of the screen. These values are used by the InitVidFrame operation to verify frame coordinates and dimensions. They are set by the ResetVideo operation.

sLine

is the total number of bytes required to contain all the information for one line of the character map. This information includes line attributes, filler bytes needed by the video hardware, text characters, and character attributes (if specified). This field can be multiplied by the number of a line to compute the offset from *pMap* of the first byte of the line. It is set by the ResetVideo operation.

defaultWindowId

is the default window identification number used in calls such as PutFrameChars and PutFrameCharsAndAttrs that do not specify a window ID.

bSpace

is the 8-bit character code that displays an empty character position on the screen. This code is 0 (null) in the standard font. The *bSpace* field of the VCB is set from a parameter to the ResetVideo operation and is used by the InitCharMap, ResetFrame, and ScrollFrame operations.

SAR

is an exact copy of the 16 bits that were last loaded into the Screen Attribute Register.

nPixelsWide

is the number of pixels wide for the bit map. If the value of *nPixelsWide* is 0, this parameter is ignored, and any horizontal resolution that can accommodate the character map can be chosen by the ResetVideoGraphics operation. (See the description of ResetVideoGraphics in Chapter 3, "Operations," for additional information.)

nPixelsHigh

is the number of pixels high for the bit map. If the value of *nPixelsHigh* is 0, this parameter is ignored, and any vertical resolution that can accommodate the character map can be chosen by the ResetVideoGraphics operation. (See the description of ResetVideoGraphics in Chapter 3, "Operations.")

wsLine

is the same as *sLine* (described above) but is a word and can, therefore, accommodate a greater value.

cLinesMaxScreen

is the maximum number of lines that can be displayed in this mode on the monitor.

fHardwareCharMap

is a flag that is TRUE if there is a hardware character map. (See the description of QueryVidHdw in Chapter 3, "Operations," for additional information.)

nPlanes

is the number of planes used for color graphics. If the value of *nPlanes* is 0, this parameter is ignored, and an appropriate number of planes is chosen for this resolution by the ResetVideoGraphics operation.

fBackgroundColor

is a flag that sets the background color mode. If *fBackgroundColor* is FALSE, all character backgrounds are color index 0. If *fBackgroundColor* is TRUE, character backgrounds are indexes 0 through 7, matching the value of the color bits in the attribute byte.

rgbRgFrame

is the array of frame descriptors, the number of which are specified at system build. *rgbRgFrame* is set by the InitVidFrame operation and cleared by the ResetVideo operation.

This page intentionally left blank.

Description

The *Voice Control Structure* provides the Telephone Service with the information it needs to perform a recording or playback. This structure is associated with the `TsVoicePlaybackFromFile` and `TsVoiceRecordToFile` operations.

See Chapter 3, "Operations," for a description of the Voice/Data services operations. For details on Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued)

Voice Control Structure

Offset	Field	Size (bytes)	Description
0	<i>fh</i>	2	is an open file handle
2	<i>lfaStart</i>	4	file position to start the recording/playback
6	<i>lfaMax</i>	4	position where record/playback terminate
10	<i>qSampleStart</i>	4	sample number to start record or playback
14	<i>qSampleMax</i>	4	number terminating recording/playback
18	<i>cPauseMax</i>	2	max silence before recording terminated
20	<i>cSampleOn</i>	2	determine the playback "fast forward" rate
22	<i>cSampleOff</i>	2	determine the playback "fast forward" rate
24	<i>f6KHz*</i>	1	determines sampling rate
25	<i>fAutoStart*</i>	1	determines record/playback start
26	<i>fNoPause*</i>	1	disables pause detection hardware
27	<i>fStopOnDialTone*</i>	1	recording terminated if dial tone detected
28	<i>fAltConnection*</i>	1	alternate connection increases gain levels
29	<i>nSectorStatusUpdate</i>	2	response to TsGetStatus requests
31	<i>sPauseGap*</i>	2	number of data bytes in pause
33	<i>fRawData*</i>	1	data not checked for escape sequences
34	<i>fPCM</i>	1	TRUE if PCM recording method used

* This field does not apply to SuperGen workstations.

fh

is the open file handle (or 0FFFFh if playing back from memory) of the voice file.

lfaStart

is the logical file address (*lfa*) at which to start recording or playing back.

lfaMax

is the *lfa* which, if reached, will terminate recording or playback.

qSampleStart

is the sample number to be assigned to the first sample recorded, or to be skipped to on playback.

qSampleMax

is the sample number, which if reached will terminate recording or playback.

cPauseMax

is the maximum silence (in units of 100ms) when recording before the recording is terminated (the terminating pause will not be included in the recording).

On playback, all pauses will be truncated to this amount. A value of 0FFFFh means no pause termination or truncation.

cSampleOn *cSampleOff*

determine the playback "fast forward" rate. A zero value for *cSampleOff* means playback at normal speed. The *cSampleOn* value determines the number of pairs of samples that will be played back, and *cSampleOff* is the number of sample pairs that will be discarded. *cSampleOn* should be greater than 50. To playback at double speed, values of 100 and 100 could be used. To playback at triple speed, values of 100 and 200 could be used.

f6KHz

is a flag that is TRUE if the CODEC sampling rate is to be 6KHz. Otherwise the rate is 8KHz (8000 4-bit samples per second).

fAutoStart

is a flag that is TRUE if recording or playing back is to start as soon as the telephone unit is placed offhook, or immediately if it is already offhook.

fNoPause

is a flag that is TRUE if the pause detection hardware is to be disabled and no pause detection or compression will be done.

fStopOnDialTone

is a flag that is TRUE if the recording is to be terminated if a dial tone is detected.

fAltConnection

is a flag that is TRUE if an alternate connection will be used (if possible) to increase gain levels.

nSectorStatusUpdate

is a value that, if greater than zero, will cause any TsGetStatus requests to be responded to each time the specified number of sectors has been processed.

sPauseGap

is the number of data bytes after start of pause and before end of pause where the actual data is used. If *sPauseGap* is 0 the default value (250) will be used.

fRawData

is a flag that is TRUE if the voice data is to be recorded or played back without examination of the data for escape sequences.

fPCM

is a flag that is TRUE if the PCM recording method is to be used instead of the ADPCM recording method. This field applies only to SuperGen workstations.

This page intentionally left blank

Description

The *Voice File Header* is a 512 byte structure associated with a voice message in a voice file. Voice File Headers are created for and used by the Voice/Data services. Using the Voice Processor's CODEC, voice messages can be recorded to the file and later played back.

The Voice File Header contains information about how the voice message was recorded, how big the file is, when the recording was made, and over which telephone lines it was recorded. Because many separate voice messages can be placed in one file, there can be multiple Voice File Headers at the beginning of the file. Each Header accomodates up to 15 messages.

For details on this structure and on how to write applications to be used with Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Voice Processor Control Block
- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice Control Structure
- Voice File Record

This page intentionally left blank

Voice File Header

Offset	Field	Size (bytes)	Description
0	signature	2	must be 'VC' (4356h)
2	version	2	version of file
4	nMessages	2	number of recordings in this file
6	rgMessages(15)	480	(see "Message Fields")
485	reserved	26	

Message Fields

Offset	Field	Size (bytes)	Description
0	f6KHz*	1	if TRUE, the data was recorded at 6KHz
1	lfaStart	4	starting lfa in file of voice data
5	lfaMax	4	ending lfa in file of voice data
9	qSampleStart	4	starting data byte count of voice data
13	qSampleMax	4	ending data byte count of voice data
17	reserved	1	
18	dateTime	4	system date/time when recording made
22	line*	1	line over which recording was made
23	fNoPause*	1	FALSE if pause compressed
24	fAltConnection*	1	TRUE if recorded with alternate connection
25	fPCM	1	TRUE if PCM recording method used
26	rgbReserved	6	

* This field does not apply to SuperGen workstations.

Voice File Header Fields

signature

must be 'VC' (4356h).

version

is the version number of the data file. Each value indicates the following:

Value	Description
1	NGEN adaptive pulse code modulation. (ADPCM)
2	SuperGen ADPCM
3	SuperGen pulse code modulation (PCM)

nMessages

is the number of messages in the file.

rgMessages

is the array of 15 messages. The format of each is described in "Message Fields," below.

Message Fields

f6KHz

is a flag that is TRUE if the data was recorded at 6KHz.

lfaStart

is the starting lfa in the file of the recording.

lfaMax

is the ending lfa in the file of the recording.

qSampleStart

is the starting data byte count of the recording.

Voice File Header

qSampleMax

is the ending data byte count of the recording.

dateTime

is the system date and time when the recording was made.

line

is the line over which the recording was made. The values of *line* are

Value	Description
0	telephone unit
1	telephone line 1
2	telephone line 2

fNoPause

is a flag that is TRUE if the pause compression was suppressed during recording.

fAltConnection

is a flag that is TRUE if the alternate (amplified) connection was used to record.

fPCM

is a flag that is TRUE if the PCM recording method is to be used instead of the ADPCM recording method. This field only applies to SuperGen workstations.

This page intentionally left blank

Description

The *Voice File Record* is the format of the voice recording sectors used by the Voice/Data services.

For details on how to write applications to be used with Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice Control Structure
- Voice File Header
- Voice Processor Control Block

This page intentionally left blank

(continued)

Voice File Record

Offset	Field	Size (bytes)	Description
0	qSampleStart	4	number of the first sample in this record
4	signature	1	always 'V' (56h)
5	version	1	version number of data file
6	rgbSample	506	voice data

qSampleStart

is the accumulated sample number of the first sample in this record.

signature

is always 'V' (56h).

version

is the version number of the data file. Each value indicates the following:

Value	Description
1	NGEN adaptive pulse code modulation (ADPCM)
2	SuperGen ADPCM
3	SuperGen pulse code modulation (PCM)

rgbSample

is the voice data bytes.

This page intentionally left blank

Voice Processor Control Block

Description

The *Voice Processor Control Block* (VPCB) is a structure in memory that the Telephone Service and the Voice Processor module use to coordinate operations.

NOTE: The information contained in this structure can be queried but normally is not modified by the application program.

For details on how to write applications to be used with Voice/Data services, see the *CTOS Programming Guide*.

Related Voice/Data Structures

- Data Control Structure
- Telephone Service Configuration Block
- Telephone Service Configuration File Format
- Telephone Status Structure
- Voice Control Structure
- Voice File Header
- Voice File Record
- Voice Processor Control Block

This page intentionally left blank

(continued)

Voice Processor Control Block

Offset	Field	Size (bytes)	Description
0	signature	2	must be 08459h
2	version	1	firmware version number
3	command	1	command to Voice Processor module
4	tUnitStatus	1	current state of the telephone unit
5	xpLine	1	device connected or data started/stopped
6	xpJuncctorMap	1	lines/terminals map
7	param0	1	parameter byte
8	param1	1	parameter byte
9	t0status	1	state of telephone line 1
10	t1status	1	state of telephone line 2
11	dialerCqcb	8	structure controlling pulse dialing
19	sDtmfGenBuf	1	number of characters to be dialed
20	pDtmfGenBuf	3	24 bit physical address of first character
23	bDtmfChar	1	character decoded using DTMF receiver
24	reserved	7	
31	modemRxCqcb	8	controls characters received over modem
39	modemTxCqcb	8	controls characters transmitted
47	codecBpcb	8	controls CODEC encoding or decoding
55	ercCodec	2	error status from CODEC
57	ercModem	2	error status from modem
59	cCptrLow	1	count of 10 ms periods of low signal
60	cCptrHigh	1	count of 10 ms periods of high signal
61	intStatus	1	interrupt status byte
62	xptMap	8	update of the cross point switch state

signature

must be the value 08459h. It is checked by the Voice Processor module firmware before generating each interrupt.

version

is the Voice Processor firmware version number.

command

is the command that the Telephone Service gives to the Voice Processor module.

tUnitStatus

is the current state of the telephone unit. The values of *tUnitStatus* are

Value	Description
0	means onhook
1	means offhook

xpLine

is the device to be connected in the crosspoint switch (XptSwitch), or for which data is to be started or stopped. The values of *xpLine* are

Value	Description
0	telephone unit (with attenuation added)
1	modem
2	DTMF generator
3	DTMF receiver
4	Call Progress Tone Receiver (CPTR) used for detecting dial tone and busy signals
5	CODEC voice digitization encoder
6	CODEC voice digitization decoder
7	telephone unit

xpJuncMap

is the bit map of telephone lines and terminals in the XptSwitch for a connection, or the telephone line for hold and hook commands, where bits are assigned as follows:

Bit	Assignment
0 (1h)	line 1
1 (2h)	line 2
2 (4h)	terminal
3 (8h)	terminal with attenuation

param0

is a parameter byte used in conjunction with various commands.

param1

is a parameter byte used in conjunction with various commands.

*t0status**t1status*

describe the current state of telephone lines 1 and 2. The values are

Value	Description
0	onhook
1	ring current detected
2	modem not ready
3	modem ready (FSK)
4	modem ready (PSK)
5	offhook

dialerCqcb

is the character queue control block (CQCB) structure used for pulse dialing.

sDtmfGenBuf

is the number of encoded characters to be dialed using the DTMF generator.

pDtmfGenBuf

is the 24 bit physical address of the first encoded character.

bDtmfChar

is the character decoded using the DTMF receiver.

modemRxCqcb

is the character queue control block (CQCB) structure for characters received over the modem.

modemTxCqcb

is the CQCB structure for characters to be transmitted over the modem.

codecBpcb

is the buffer pointer control block (BPCB) for CODEC encoding or decoding.

ercCodec

is the error status from the CODEC.

ercModem

is the error status from the modem.

(continued)

Voice Processor Control Block

cCptrLow

cCptrHigh

are the count of 10 ms periods of low and high signals detected by the CPTR. The low/high values and their meanings are

Value (Low/High)	Description
0/255	dial tone
45–55/45–55	busy
15–35/15–35	fast busy

Other values are not defined and may mean no connection, voice, data, or noise on the line.

intStatus

is a status byte placed here by the Voice Processor module firmware before generating the XINT4 interrupt signal to wake up the Telephone Service. The Telephone Service places the value 0FFh here after processing the interrupt.

xptMap

is the state of the crosspoint switch.

This page intentionally left blank

Caution: *This system structure is for restricted use only and is subject to change in future releases of the operating system.*

Description

The *Volume Home Block (VHB)* is the root structure (that is, the starting point for the tree structure) of the information that describes a file system on a disk volume. Information in the VHB includes, at a minimum, the volume name and the date it was created, and the memory addresses of the Allocation Bit Map, the Bad Sector File (physically addressed devices only), the File Header Blocks, the Master File Directory, the System Image, the Crash Dump Area, and the Log File. (For a description and illustration of the VHB and its relation to the other volume control structures, see "File Management," in the *CTOS Operating System Concepts Manual*.)

All of the volume control structures are created when a disk volume is initialized using the **Format Disk** command. (For details on **Format Disk**, see the *CTOS Executive Reference Manual*.) For enhanced reliability, **Format Disk** creates two copies of the VHB: one within the first 15 sectors, and a second near the middle of the disk.

The operating system brings the VHB into memory when a volume is mounted. When any file system operation (such as **OpenFile** or **CreateFile**) is called, the operating system verifies that the volume of the requested name is present by checking for the VHB in memory.

The size of the VHB is 256 bytes but it is kept on the medium in a block (512 bytes). The second half of the VHB block is not used and is reserved for future expansion.

The VHB can describe both physically addressed devices (sometimes referred to as *CHS devices*) and logically addressed devices (also called *LBA* or *logical block addressable devices*). Physically addressed devices represent an earlier generation of direct-access devices that did not have "intelligence" embedded within the disk controller. That is, the VHB had to explicitly describe the size and geometry of the disk platters in terms of cylinders per disk, tracks (heads) per cylinder, physical sectors per track and bytes per physical sector. In contrast, logically addressed devices typically contain controller intelligence within the device that hides physical details of the disk organization and presents the appearance of a contiguous address space from block zero up to the maximum capacity of the disk. SCSI disks are one example of logically addressed devices; all future generations of direct-access devices are likely to be logically addressable. Logically addressed devices typically perform their own medium defect management schemes and support features that allow their operating characteristics and limitations to be determined dynamically by the computer systems that connect to them.

By default, an entry in the configuration file specifies that a disk be formatted to describe a physically addressed device. The default entry has the following format:

:OldCTOSFormat?: yes

(See the description of the **Format Disk** command in the *CTOS Executive Reference Manual*.) If, however, the keyword entry is not present in the configuration file, the default way the disk is formatted depends on the operating system version, as follows:

OS Version	Format
CTOS/XE 3.0	LBA device
CTOS I 3.3	Physically addressed device
CTOS II 3.3	LBA device

The VHB used on earlier operating system versions described physically addressed devices, only. In order for some logically addressed devices to be supported, *e.g.* workstation SCSI disk modules, a fictitious disk geometry of cylinders, heads and sectors was invented for each disk and placed into the VHB. When logically addressed devices are described by the new style VHB, any fields that are meant to describe physically addressed devices are filled with zeros.

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Entry for a Directory in a Master File Directory Block
- Entry for a File in a Directory Block
- Master File Directory Block
- File Header Block
- Volume Home Block (pre-3.0)

This page intentionally left blank

(continued)

Volume Home Block

Offset	Field	Size (bytes)	Description
0	checksum	2	value used to verify the integrity of the VHB
2	lfaSysImageBase	4	disk address of the System Image
6	cPagesSysImage	2	number of blocks in the System Image
8	lfaBadBlkBase	4	disk address of the Bad Sector file
12	cPagesBadBlk	2	number of blocks in the Bad Sector file
14	lfaCrashDumpBase	4	disk address of the Crash Dump file
18	cPagesCrashDump	2	number of blocks in the Crash Dump file
20	sbVolName	13	volume name
33	sbVolPassword	13	volume password
46	lfaVhb	4	disk address of the "working" VHB
50	lfaInitialVhb	4	disk address of the backup VHB
54	creationDT	4	date the volume was created
58	modificationDT	4	date any file was modified
62	lfaMfdBase	4	disk address of the master file directory
66	cPagesMfd	2	number of blocks in the MFD
68	lfaLogBase	4	disk address of system Log file
72	cPageLog	2	number of disk blocks in the Log file
74	currentLogPage	2	current block number to be written to
76	currentLogByte	2	next byte to write to in <i>currentLogPage</i>
78	lfaFileHeadersBase	4	disk address of the File Header blocks
82	cPagesFileHeader	2	number of blocks in File Header blocks
84	altFileHeaderPageOffset	2	used to find secondary File Header
86	iFreeFileHeader	2	index of first free File Header block
88	cFreeFileHeaders	2	number of free File Header blocks
90	reserved	1	
91	bECC	1	volume formatting in ECC mode
92	reserved	4	
96	lfaAllocBase	4	disk address of Allocation Bit Map
100	allocPageCnt	2	number of blocks in Allocation Bit Map
102	lastAllocPg	2	index of last updated block
104	lastAllocWd	2	index of last word updated in the block
106	lastAllocBit	2	index of last bit updated in the word
108	cFreePages	4	number of unused blocks
112	iDev	2	not used
114	seekStepRate	1	interval between seek step pulses
115	trackSeekTime	1	average seek time for one track
116	settleTime	1	time to wait after a seek for head to settle
117	gapLength	1	length of unused area between sectors
118	writePrecompCyl	1	write precompensation starting cylinder

(continued)

Volume Home Block

(continued)

119	devType	1	(internal use) tells if volume is removable
120	spiralFactor	1	for SMD drives
121	startingSector	1	for SMD drives
122	interleaveFactor	1	for SMD drives
123	verifyCode	1	number of server reboots
124	reserved	95	
219	magicWd	2	indicates the type of VHB
221	bootBaseSector	1	sector at which System Image is located
222	bootBaseHead	1	head at which System Image is located
223	bootBaseCyl	2	cylinder at which System Image is located
225	bootMaxPageCount	2	number of blocks in the System Image
227	badBlkBaseSector	1	sector at which Bad Sector file is located
228	badBlkBaseHead	1	head at which Bad Sector file is located
229	badBlkBaseCyl	2	cylinder at which Bad Sector file is located
231	badBlkMaxPgCnt	2	number of blocks in the Bad Sector file
233	dumpBaseSector	1	sector at which Crash Dump file is located
234	dumpBaseHead	1	head at which Crash Dump file is located
235	dumpBaseCyl	2	cylinder where Crash Dump file is located
237	dumpMaxPageCount	2	number of blocks in the Crash Dump file
239	bytesPerSector	2	bytes per physical sector
241	sectorsPerTrack	2	physical sectors per track
243	tracksPerCyl	2	tracks per cylinder
245	cylindersPerDisk	2	cylinders per disk
247	interfaceClass	1	type of hardware programmatic interface
248	volumeCapacity	4	number of user-accessible logical blocks
252	deviceClass	1	device family, such as direct access
253	reserved	3	

checksum

is an internal value used to verify the integrity of the VHB. If the sum of the numeric values of all the other words in the VHB do not equal this value, the File System will stop working and a status code or system crash may be generated.

lfaSysImageBase

is the disk address of the System Image file <Sys>SysImage.sys.

cPagesSysImage

is the number of disk blocks in <Sys>SysImage.sys.

lfaBadBlkBase

is the disk address of the file <Sys>BadBlock.sys. This field should be 0 for devices that perform their own defect management.

cPagesBadBlk

is the number of disk blocks in <Sys>BadBlock.sys. This field should be 0 for devices that perform their own defect management.

lfaCrashDumpBase

is the disk address of the system Crash Dump file <Sys>CrashDump.sys.

cPagesCrashDump

is the number of disk blocks in <Sys>CrashDump.sys.

sbVolName

is the volume name.

sbVolPassword

is the volume password.

lfaVhb

is the disk address of the "working" copy of the Volume Home Block.

lfaInitialVhb

is the disk address of the backup copy of the Volume Home Block.

creationDT

is the date and time the volume was created.

modificationDT

is the date and time any file was last modified (created, deleted, or had its length changed).

lfaMfdBase

is the disk address of the Master File Directory.

cPagesMfd

is the number of disk blocks taken up by the Master File Directory.

lfaLogBase

is the disk location where the Log file is placed when the file is created. This field is only important to the file system, which has many requests to access the disk and needs to make these requests as rapidly as possible.

cPageLog

is the number of disk blocks in the Log file.

currentLogPage

is the number of the current block to be written to (a value in the range 0 through *cPageLog*-1).

currentLogByte

is the next byte to be written to in the current block (a value in the range 0 through 511).

lfaFileHeadersBase

is the disk address of the File Header blocks.

cPagesFileHeader

is the number of blocks in the File Header blocks.

altFileHeaderPageOffset

is a number which is added to the block location of the primary File Header block to find the secondary File Header block.

iFreeFileHeader

is the index of the first free File Header block.

cFreeFileHeaders

is the number of free File Header blocks.

bEcc

indicates whether or not the volume was formatted in ECC mode.

lfaAllocBase

is the disk address of the Allocation Bit Map.

allocPageCnt

is the number of blocks in the Allocation Bit Map.

lastAllocPg

is the index of the last block updated in the Allocation Bit Map.

lastAllocWd

is the index of the last word updated within the last block updated in the Allocation Bit Map.

lastAllocBit

is the index of the last bit updated within the last word updated in the Allocation Bit Map.

cFreePages

is the number of unused blocks.

iDev

is not used.

seekStepRate

is an encoded number that represents the time interval between seek step pulses for disks that are controlled by a WD-1010 or WD-2010.

gapLength

is a parameter value to the device controller that indicates the length of the unused area between sectors of a formatted disk.

writePrecompCyl

is the cylinder where write precompensation starts.

devType

is internally used by the file system to determine if the volume is removable.

spiralFactor

is the sector number offset between adjacent tracks for disks (typically SMD) that use a spiral layout of data sectors to improve performance.

startingSector

is the first sector Id on a track.

interleaveFactor

is the interval of sequential sector access by the disk hardware controller, for example, every other sector or every third sector. Disk accesses that skip over sectors in this way allow time for buffers to be emptied and ready to accept data without rotational delay.

verifyCode

is a value that is incremented each time the system is rebooted. This value wraps around to zero.

magicWd

indicates the type of VHB. The values of *magicWd* are

Value	Description
31801	pre-3.0 VHB without volume encryption
20547	pre-3.0 VHB with volume encryption
Value	Description
32768 through 65535	VHB for CTOS/XE 3.0 or higher operating systems. The remaining bits of <i>magicWd</i> for this VHB are as follows:

Bit	Meaning when set	Meaning when clear
15*	This is a CTOS/XE 3.0 or higher VHB	
14	Logically addressed device	Physically addressed device
1	Password encryption enabled	Password encryption disabled
0	Hierarchical file system	Flat file system

*This bit is always set.

*bootBaseSector***

is the sector at which the System Image is located.

*bootBaseHead***

is the head at which the System Image is located.

*bootBaseCyl***

is the cylinder at which the System Image is located.

*bootMaxPageCount***

is the number of blocks in the System Image.

*badBlkBaseSector***

is the sector at which the Bad Sector file is located.

*badBlkBaseHead***

is the head at which the Bad Sector file is located.

*badBlkBaseCyl***

is the cylinder at which the Bad Sector file is located.

*badBlkMaxPgCnt***

is the number of blocks in the Bad Sector file.

*dumpBaseSector***

is the sector at which the Crash Dump file is located.

****These fields are non-zero only if the VHB is for a physically addressed device. They are a translation of the disk address of the file into cylinder, head and sector form.**

*dumpBaseHead***

is the head at which the Crash Dump file is located.

*dumpBaseCyl***

is the cylinder at which the Crash Dump file is located.

*dumpMaxPageCount***

is the number of blocks in the Crash Dump file.

bytesPerSector

is a volume capacity parameter indicating the number of bytes per physical sector.

sectorsPerTrack

is a volume capacity parameter indicating the number of sectors per track.

tracksPerCyl

is a volume capacity parameter indicating the number of tracks (heads) per cylinder.

cylindersPerDisk

is a volume capacity parameter indicating the number of cylinders per disk.

****These fields are non-zero only if the VHB is for a physically addressed device. They are a translation of the disk address of the file into cylinder, head and sector form.**

interfaceClass

is a value that indicates the hardware interface used by the device. The value ranges are

Value range	Description
0 to 127	reserved for hardware programmatic interfaces that require the operating system to know the device operating characteristics in great detail.
128 to 255	reserved for intelligent hardware programmatic interfaces.

volumeCapacity

for physically addressed devices, this is the product of the fields *cylindersPerDisk*, *tracksPerCyl*, *sectorsPerTrack* and *bytesPerSector*. For logically addressed devices this is usually obtained from the device itself (e.g. the READ CAPACITY command is used for SCSI disks).

deviceClass

is the family to which the device belongs, such as direct address, sequential, or WORM. At present it is not used and is provided for future expansion.

Volume Home Block (pre-3.0)

Caution: *This system structure is for restricted use only and is changed for future releases of the operating system.*

Description

The *Volume Home Block (pre-3.0)* is used by programs to format physical address devices on operating systems prior to CTOS/XE 3.0. (See the description of CurrentOsVersion in Chapter 3, "Operations." The pre-3.0 operating systems have internal version numbers less than 12.0.)

The Volume Home Block (VHB) is the root structure (that is, the starting point for the tree structure) of the information on a disk volume. (For a detailed description of the VHB, see "Volume Home Block," earlier in this chapter.)

Related File Structures

- Device Control Block
- Device Control Block (pre-3.0)
- Entry for a Directory in a Master File Directory Block
- Entry for a File in a Directory Block
- File Header Block
- Master File Directory Block
- Volume Home Block

This page intentionally left blank

(continued)

Volume Home Block (pre-3.0)

Offset	Field	Size (bytes)	Description
0	checksum	2	value used to verify the integrity of the VHB
2	lfaSysImageBase	4	disk address of the System Image
6	cPagesSysImage	2	number of blocks in the System Image
8	lfaBadBlkBase	4	disk address of the Bad Sector file
12	cPagesBadBlk	2	number of blocks in the Bad Sector file
14	lfaCrashDumpBase	4	disk address of the Crash Dump file
18	cPagesCrashDump	2	number of blocks in the Crash Dump file
20	sbVolName	13	volume name
33	sbVolPassword	13	volume password
46	lfaVhb	4	disk address of the "working" VHB
50	lfaInitialVhb	4	disk address of the backup VHB
54	creationDT	4	date the volume was created
58	modificationDT	4	date any file was modified
62	lfaMfdBase	4	disk address of the master file directory
66	cPagesMfd	2	number of blocks in the MFD
68	lfaLogBase	4	disk address of system Log file
72	cPageLog	2	number of disk blocks in the Log file
74	currentLogPage	2	current block number to be written to
76	currentLogByte	2	next byte to write to in <i>currentLogPage</i>
78	lfaFileHeadersBase	4	disk address of the File Header blocks
82	cPagesFileHeader	2	number of blocks in File Header blocks
84	altFileHeaderPageOffset	2	used to find secondary File Header
86	iFreeFileHeader	2	index of first free File Header block
88	cFreeFileHeaders	2	number of free File Header blocks
90	reserved	6	
96	lfaAllocBase	4	disk address of Allocation Bit Map
100	allocPageCnt	2	number of blocks in Allocation Bit Map
102	lastAllocPg	2	index of last updated block
104	lastAllocWd	2	index of last word updated in the block
106	lastAllocBit	2	index of last bit updated in the word
108	cFreePages	4	number of unused blocks
112	iDev	2	DCB index when volume is mounted
114	reserved	105	
219	magicWd	2	indicates the type of VHB
221	bootBaseSector	1	disk address of the System Image
222	bootBaseHead	1	head on which System Image is located
223	bootBaseCyl	2	cylinder on which System Image is located
225	bootMaxPageCount	2	number of blocks in the System Image

(continued)

Volume Home Block (pre-3.0)

(continued)

227	badBlkBaseSector	1	disk address of the Bad Sector file
228	badBlkBaseHead	1	head on which Bad Sector file is located
229	badBlkBaseCyl	2	cylinder on which Bad Sector file is located
231	badBlkMaxPgCnt	2	disk address of the Crash Dump file
233	dumpBaseSector	1	disk address of the Crash Dump file
234	dumpBaseHead	1	head on which Crash Dump file is located
235	dumpBaseCyl	2	cylinder the Crash Dump file is on
237	dumpMaxPageCount	2	number of blocks in the Crash Dump file
239	bytesPerSector	2	bytes per disk sector
241	sectorsPerTrack	2	sectors per track
243	tracksPerCyl	2	tracks per cylinder
245	cylindersPerDisk	2	cylinders per disk

WORKSTATIONS ONLY (OFFSETS 247 through 253)

247	verifyCode	1	number of server reboots
248	sectorSize	2	sector size
250	spiralFactor	1	for SMD drives
251	startingSector	1	for SMD drives
252	interleaveFactor	1	for SMD drives
253	vendorCode (3)	3	disk manufacture and physical geometry

SHARED RESOURCE PROCESSOR ONLY (OFFSETS 247 through 253)

247	seekStepRate	1	time interval between seek step pulses
248	gapSize	1	length of unused area between sectors
249	writePrecompCyl	1	start cylinder for write precomp current
250	devType	1	determines if volume is removable
251	trackSeekTime	1	average seek time for one track
252	settleTime	1	time to wait after a seek for head to settle
253	verifyCode	1	number of server reboots

checksum

is an internal value used to verify the integrity of the VHB. If the sum of the numeric values of all the other words in the VHB do not equal this value, the File System will stop working and a status code or system crash may be generated.

lfaSysImageBase

is the disk address of the System Image file <Sys>SysImage.sys.

cPagesSysImage

is the number of disk blocks in <Sys>SysImage.sys.

lfaBadBlkBase

is the disk address of the file <Sys>BadBlock.sys. This field should be 0 for devices that perform their own defect management.

cPagesBadBlk

is the number of disk blocks in <Sys>BadBlock.sys. This field should be 0 for devices that perform their own defect management.

lfaCrashDumpBase

is the disk address of the system Crash Dump file <Sys>CrashDump.sys.

cPagesCrashDump

is the number of disk blocks in <Sys>CrashDump.sys.

sbVolName

is the volume name.

sbVolPassword

is the volume password.

lfaVhb

is the disk starting address of the "working" copy of the Volume Home Block.

lfaInitialVhb

is the disk address of the backup copy of the Volume Home Block.

creationDT

is the date and time the volume was created.

modificationDT

is the date and time any file was last modified (created, deleted, or had its length changed).

lfaMfdBase

is the disk address of the Master File Directory.

cPagesMfd

is the number of disk blocks taken up by the Master File Directory.

lfaLogBase

is the disk location where the Log file is placed when the file is created. This field is only important to the file system, which has many requests to access the disk and needs to make these requests as rapidly as possible.

cPageLog

is the number of disk blocks in the Log file.

currentLogPage

is the number of the current block to be written to (a value in the range 0 through *cPageLog*-1).

currentLogByte

is the next byte to be written to in the current block (a value in the range 0 through 511).

lfaFileHeadersBase

is the disk address of the File Header blocks.

cPagesFileHeader

is the number of blocks in the File Header blocks.

altFileHeaderPageOffset

is a number which is added to the block location of the primary File Header block to find the secondary File Header block.

iFreeFileHeader

is the index of the first free File Header block.

cFreeFileHeaders

is the number of free File Header blocks.

lfaAllocBase

is the disk address of the Allocation Bit Map.

allocPageCnt

is the number of blocks in the Allocation Bit Map.

lastAllocPg

is the index of the last block updated in the Allocation Bit Map.

lastAllocWd

is the index of the last word updated within the last block updated in the Allocation Bit Map.

lastAllocBit

is the index of the last word updated within the last block updated in the Allocation Bit Map.

cFreePages

is the number of unused blocks.

iDev

gives DCB index when volume is mounted.

magicWd

indicates the type of VHB. The values of *magicWd* are

Value	Description
31801	pre-3.0 VHB without volume encryption
20547	pre-3.0 VHB with volume encryption

bootBaseSector

is the sector at which the System Image is located.

bootBaseHead

is the head at which the System Image is located.

bootBaseCyl

is the cylinder at which the System Image is located.

bootMaxPageCount

is the number of blocks in the System Image.

badBlkBaseSector

is the sector at which the Bad Sector file is located.

badBlkBaseHead

is the head at which the Bad Sector file is located.

badBlkBaseCyl

is the cylinder at which the Bad Sector file is located.

badBlkMaxPgCnt

is the number of blocks in the Bad Sector file.

dumpBaseSector

is the sector at which the Crash Dump file is located.

dumpBaseHead

is the head at which the Crash Dump file is located.

dumpBaseCyl

is the cylinder at which the Crash Dump file is located.

dumpMaxPageCount

is the number of blocks in the Crash Dump file.

bytesPerSector

is a volume capacity parameter indicating the number of bytes per physical sector.

sectorsPerTrack

is a volume capacity parameter indicating the number of sectors per track.

tracksPerCyl

is a volume capacity parameter indicating the number of tracks per cylinder.

cylindersPerDisk

is a volume capacity parameter indicating the number of cylinders per disk.

WORKSTATIONS ONLY (OFFSETS 247 through 253)*verifyCode*

is the number of times the server has been rebooted.

sectorSize

is a value that encodes the physical sector size. The size can be one of the following values:

Value	Sector size
0	use bytes per sector field (must be less than 256)
1	256 bytes
2	512 bytes
3	1024 bytes

spiralFactor

is the sector number offset between adjacent tracks for disks (typically SMD) that use a spiral layout of data sectors to improve performance.

startingSector

is the first sector Id on a track.

interleaveFactor

is the interval of sequential sector access by the disk hardware controller, for example, every other sector or every third sector. Disk accesses that skip over sectors in this way allow time for buffers to be emptied and ready to accept data without rotational delay.

vendorCode

is an alpha value that indicates the manufacturer and disk geometry (number of cylinders and heads) of a disk (ST-506 devices only).

SHARED RESOURCE PROCESSOR ONLY (OFFSETS 247 through 253)

seekStepRate

is an encoded number that represents the time interval between seek step pulses for disks that are controlled by a WD-1010 or WD-2010.

gapSize

is the length of the unused area between sectors of a formatted disk.

writePrecompCyl

is the cylinder at which write precompensation starts.

devType

is used internally by the file system to determine if a volume is removable.

trackSeekTime

is the average seek time for one track.

settleTime

is the time to wait after a seek for head to settle.

verifyCode

is the number of times the server has been rebooted.

This page intentionally left blank

A

Status Codes

See the *CTOS Status Codes Reference Manual*.

B

Standard Character Set

Table B-1 describes the standard 256-entry character set used when the keyboard is in character mode. This character set is derived from the Translation Data Block in the standard version of NlsKbd.sys. The standard font for the character set resides in the font RAM. Table B-2 below shows the graphic representation of the characters of Table B-1.

Code Keys

When the keyboard is in character mode, the two Code keys are special kinds of Shift keys. If either or both is depressed when a non-Shift key is pressed, the high-order bit of the key is turned on. For example, Code-A generates $80h + 61h = 0E1h$, Code-space generates $80h + 20h = 0A0h$, and so forth. Note that any of the values $80h...0FFh$ can be generated from the keyboard in this way.

In addition, some of the character codes in the range $80h$ to $0DFh$ have keyboard encodings that do not require the Code key.

Legend for Table B-1

Where a character can be generated only by pressing Shift and/or Code and another key, the key combination is shown as a hyphenated list of keys (for example, Shift-6).

The four empty key posts covered by the double keys left-Shift, right-Shift, numeric-0, and Next are denoted by (SH-L'), (SH-R'), (0'), and (Next'), respectively.

The keys on the numeric pad are denoted "num 0", etc., to distinguish them from the corresponding keys on the typewriter pad.

Table B-1. Standard Character Set
(Page 1 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
00h	null	Help
01h	(up arrow) see Table B-2	Up arrow
02h	(triangle) see Table B-2	Mark
03h	¢	Shift-6
04h	(filled square) see Table B-2	Finish
05h	□ (empty square)	Prev Page
06h	1/2	1/2
07h	bell	Cancel
08h	backspace	Backspace
09h	tab	Tab
0Ah	new line	Return Next
0Bh	(down arrow) see Table B-2	Down arrow
0Ch	formfeed	Next Page
0Dh	(triangle) see Table B-2	Bound
0Eh	(left arrow) see Table B-2	Left arrow
0Fh	‡ (double dagger)	Move
10h	1/4	Shift-1/2
11h	†	Scroll Up
12h	(right arrow) see Table B-2	Right arrow
13h	trough	Scroll Down
14h	raised dot	Copy
15h	see Table B-2	F1
16h	(vertical bar) see Table B-2	F2
17h	§	F3

Table B-1. Standard Character Set
(Page 2 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
18h	see Table B-2	F4
19h	similarity	F5
1Ah	¶	F6
1Bh	● (bullet)	Go
1Ch	not	F7
1Dh	see Table B-2	F8
1Eh	see Table B-2	F9
1Fh	see Table B-2	F10
20h	space	Space
21h	!	Shift-1
22h	"	Shift-'
23h	#	Shift-3
24h	\$	Shift-4
25h	%	Shift-5
26h	&	Shift-7
27h	' (single quote)	,
28h	(Shift-9
29h)	Shift-0
2Ah	*	Shift-8
2Bh	+	Shift-=
2Ch	, (comma)	,
2Dh	- (hyphen)	-
2Eh	. (period)	.
2Fh	/	/
30h	0	0

Table B-1. Standard Character Set
(Page 3 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
31h	1	1
32h	2	2
33h	3	3
34h	4	4
35h	5	5
36h	6	6
37h	7	7
37h	8	8
39h	9	9
3Ah	:	Shift-;
3Bh	;	;
3Ch	<	Shift-[
3Dh	=	=
3Eh	>	Shift-]
3Fh	?	Shift-/
40h	@	Shift-2
41h	A	Shift-a
42h	B	Shift-b
43h	C	Shift-c
44h	D	Shift-d
45h	E	Shift-e
46h	F	Shift-f
47h	G	Shift-g
48h	H	Shift-h
49h	I	Shift-i

Table B-1. Standard Character Set
(Page 4 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
4Ah	J	Shift-j
4Bh	K	Shift-k
4Ch	L	Shift-l
4Dh	M	Shift-m
4Eh	N	Shift-n
4Fh	O	Shift-o
50h	P	Shift-p
51h	Q	Shift-q
52h	R	Shift-r
53h	S	Shift-s
54h	T	Shift-t
55h	U	Shift-u
56h	V	Shift-v
57h	W	Shift-w
58h	X	Shift-x
59h	Y	Shift-y
5Ah	Z	Shift-z
5Bh	[[
5Ch	\ (backslash)	Shift-num 8
5Dh]]
5Eh	^ (caret)	^
5Fh	_ (underline)	Shift-- (hyphen)
60h	' (reverse accent)	Shift-num 1
61h	a	a
62h	b	b

Table B-1. Standard Character Set
(Page 5 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
63h	c	c
64h	d	d
65h	e	e
66h	f	f
67h	g	g
68h	h	h
69h	i	i
6Ah	j	j
6Bh	k	k
6Ch	l	l
6Dh	m	m
6Eh	n	n
6Fh	o	o
70h	p	p
71h	q	q
72h	r	r
73h	s	s
74h	t	t
75h	u	u
76h	v	v
77h	w	w
78h	x	x
79h	y	y
7Ah	z	z

Table B-1. Standard Character Set
(Page 6 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
7Bh	{	Shift-num 4
7Ch	see Table B-2	Shift-num 7
7Dh	}	Shift-num 5
7Eh	~ (tilde)	Shift-^
7Fh	filled rectangle	Delete
80h	null	Code-Help (SH-L')
81h		Code-Up arrow Shift (SH-L')
82h	₀	Code-Mark (SH-R')
83h	₁	Code-Shift-6 Shift (SH-R')
84h	₂	Code-Finish (0')
85h	₃	Code-Prev Page Shift (0')
86h	₄	Code-1/2 (Next')
87h	₅	Code-Cancel Shift (Next')
88h	₆	Code-Backspace
89h	₇	Code-Tab
8Ah	₈	Code-Return Code-Next
8Bh	₉	Code-Down arrow
8Ch	⁰ (superscript)	Code-Next Page
8Dh	¹ (superscript)	Code-Bound
8Eh	² (superscript)	Code-Left arrow
8Fh	³ (superscript)	Code-Move
90h	⁴ (superscript)	Code-Shift-1/2

Table B-1. Standard Character Set
(Page 7 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
91h	5 (superscript)	Code-Scroll Up
92h	6 (superscript)	Code-Right arrow
93h	7 (superscript)	Code-Scroll Down
94h	8 (superscript)	Code-Copy
95h	9 (superscript)	Code-f1
96h	0 (subscript)	Code-f2
97h	1 (subscript)	Code-f3
98h	2 (subscript)	Code-f4
99h	3 (subscript)	Code-f5
9Ah	4 (subscript)	Code-f6
9Bh	5 (subscript)	Code-Go
9Ch	6 (subscript)	Code-f7
9Dh	7 (subscript)	Code-f8
9Eh	8 (subscript)	Code-f9
9Fh	9 (subscript)	Code-f10
A0h*	A circle	Code-Space
A1h*	a circle	Code-Shift-1
A2h*	A umlaut	Code-Shift-'
A3h*	a umlaut	Code-Shift-3
A4h*	O umlaut	Code-Shift-4
A5h*	o umlaut	Code-Shift-5
A6h*	O slashed	Code-Shift-7
A7h*	o slashed	Code-'
A8h*	U umlaut	Code-Shift-9
A9h*	u umlaut	Code-Shift-0

*See Table B-3 for nationalized characters.

Table B-1. Standard Character Set
(Page 8 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
AAh*	c cedilla	Code-Shift-8
ABh*	e circumflex	Code-Shift-=
ACh*	e grave	Code-,
ADh*	e acute	Code-- (hyphen)
AEnh*	AE diphthong	Code-.
AFh*	ae diphthong	Code-/
B0h*	ø	Code-0
B1h*	£	Code-1
B2h*	° (degree)	Code-2
B3h*	©	Code-3
B4h*	®	Code-4
B5h*	™	Code-5
B6h*	l	Code-6
B7h*	l ₁	Code-7
B8h*	l ₂	Code-8
B9h*	l ₃	Code-9
BAh*	l ₄	Code-Shift-;
BBh*	l ₅	Code-;
BCh*	l ₆	Code-Shift-[
BDh*	l ₇	Code-=
BEh*	l ₈	Code-Shift-]
BFh*	l ₉	Code-Shift-/
COh	see Table B-2	Code-Shift-2 Shift-Help

*See Table B-3 for nationalized characters.

Table B-1. Standard Character Set
(Page 9 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
C1h	see Table B-2	Code-Shift-a Shift-Up arrow
C2h	see Table B-2	Code-Shift-b Shift-Mark
C3h	see Table B-2	Code-Shift-c Shift-Bound
C4h	see Table B-2	Code-Shift-d Shift-Finish
C5h	see Table B-2	Code-Shift-e Shift-Prev Page
C6h	see Table B-2	Code-Shift-f
C7h	see Table B-2	Code-Shift-g Shift-Cancel
C8h	see Table B-2	Code-Shift-h Shift-Delete
C9h	see Table B-2	Code-Shift-i Shift-Go
CAh	see Table B-2	Code-Shift-j Shift-F9
CBh	see Table B-2	Code-Shift-k Shift-Down arrow
CCh	see Table B-2	Code-Shift-l Shift-Next Page
CDh	see Table B-2	Code-Shift-m Shift-F8
CEh	see Table B-2	Code-Shift-n Shift-Left arrow
CFh	see Table B-2	Code-Shift-o Shift-Move
D0h	see Table B-2	Code-Shift-p Overtyp

Table B-1. Standard Character Set
(Page 10 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
D1h	see Table B-2	Code-Shift-q Shift-Scroll Up
D2h	see Table B-2	Code-Shift-r Shift-Right arrow
D3h	see Table B-2	Code-Shift-s Shift-Scroll Down
D4h	see Table B-2	Code-Shift-t Shift-Copy
D5h	see Table B-2	Code-Shift-u Shift-F1
D6h	see Table B-2	Code-Shift-v Shift-F2
D7h	see Table B-2	Code-Shift-w Shift-F3
D8h	see Table B-2	Code-Shift-x Shift-F4
D9h	see Table B-2	Code-Shift-y Shift-F5
DAh	see Table B-2	Code-Shift-z Shift-F6
DBh	see Table B-2	Code-[
DCh	see Table B-2	Code-Shift-num 8 Shift-F7
DDh	see Table B-2	Code-]
DEh	see Table B-2	Code-^
DFh	see Table B-2	Code-Shift-- (hyphen) Shift-F10
E0h	see Table B-2	Code-Shift-num 1
E1h	see Table B-2	Code-a
E2h	see Table B-2	Code-b
E3h	see Table B-2	Code-c
E4h	see Table B-2	Code-d

Table B-1. Standard Character Set
(Page 11 of 11)

Character Code	Video Display Character	Literal Keystroke(s)
E5h	see Table B-2	Code-e
E6h	see Table B-2	Code-f
E7h	see Table B-2	Code-g
E8h	see Table B-2	Code-h
E9h	see Table B-2	Code-i
EAh	see Table B-2	Code-j
EBh	see Table B-2	Code-k
ECh	see Table B-2	Code-l
EDh	see Table B-2	Code-m
EEh	see Table B-2	Code-n
EFh	see Table B-2	Code-o
F0h	see Table B-2	Code-p
F1h	see Table B-2	Code-q
F2h	see Table B-2	Code-r
F3h	see Table B-2	Code-s
F4h	see Table B-2	Code-t
F5h	see Table B-2	Code-u
F6h	see Table B-2	Code-v
F7h	see Table B-2	Code-w
F8h	see Table B-2	Code-y
F9h	see Table B-2	Code-x
FAh	see Table B-2	Code-z
FBh	see Table B-2	Code-Shift-num 4
FCh	see Table B-2	Code-Shift-num 7
FDh	bar chart	Code-Shift-num 5
FEh	bar chart	Code-Shift-^
FFh	bar chart	Code-Delete

**Table B-2. Graphic Representation of the
Standard Character Set**

(Page 1 of 2)

Character Code	Video Display Character	Character Code	Video Display Character	Character Code	Video Display Character	Character Code	Video Display Character
00h		20h		40h	@	60h	'
01h	↑	21h	!	41h	A	61h	a
02h	▴	22h	"	42h	B	62h	b
03h	¢	23h	#	43h	C	63h	c
04h	■	24h	\$	44h	D	64h	d
05h	□	25h	%	45h	E	65h	e
06h	½	26h	&	46h	F	66h	f
07h	↓	27h	'	47h	G	67h	g
08h	←	28h	(48h	H	68h	h
09h	→	29h)	49h	I	69h	i
0Ah	√	2Ah	*	4Ah	J	6Ah	j
0Bh	↓	2Bh	+	4Bh	K	6Bh	k
0Ch	↘	2Ch	.	4Ch	L	6Ch	l
0Dh	↗	2Dh	—	4Dh	M	6Dh	m
0Eh	←	2Eh	.	4Eh	N	6Eh	n
0Fh	±	2Fh	/	4Fh	O	6Fh	o
10h	¼	30h	0	50h	P	70h	p
11h	½	31h	1	51h	Q	71h	q
12h	→	32h	2	52h	R	72h	r
13h	↪	33h	3	53h	S	73h	s
14h		34h	4	54h	T	74h	t
15h	÷	35h	5	55h	U	75h	u
16h	ı	36h	6	56h	V	76h	v
17h	\$	37h	7	57h	W	77h	w
18h	≠	38h	8	58h	X	78h	x
19h	≈	39h	9	59h	Y	79h	y
1Ah	¶	3Ah	:	5Ah	Z	7Ah	z
1Bh	●	3Bh	;	5Bh	[7Bh	{
1Ch	¬	3Ch	<	5Ch	\	7Ch	
1Dh	≤	3Dh	=	5Dh]	7Dh	}
1Eh	±	3Eh	>	5Eh	^	7Eh	~
1Fh	≥	3Fh	?	5Fh	—	7Fh	■

**Table B-2. Graphic Representation of the
Standard Character Set**

(Page 2 of 2)

Character Code	Video Display Character	Character Code	Video Display Character	Character Code	Video Display Character	Character Code	Video Display Character
80h		A0h	À	C0h	┆	E0h	ℒ
81h		A1h	Á	C1h	┆	E1h	
82h	0	A2h	Ä	C2h	┆	E2h	┆
83h	1	A3h	ä	C3h	+	E3h	┆
84h	2	A4h	Ö	C4h	┆	E4h	┆
85h	3	A5h	ö	C5h	┆	E5h	┆
86h	4	A6h	Ø	C6h	┆	E6h	┆
87h	5	A7h	ø	C7h	+	E7h	┆
88h	6	A8h	Û	C8h	+	E8h	┆
89h	7	A9h	ü	C9h	┆	E9h	┆
8Ah	8	AAh	ç	CAh	┆	EAh	┆
8Bh	9	ABh	ê	CBh	+	EBh	┆
8Ch	0	ACH	é	CCh	┆	ECh	┆
8Dh	1	ADh	è	CDh	┆	EDh	┆
8Eh	2	AEnh	æ	CEh	┆	EEh	┆
8Fh	3	AFh	æ	CFh	+	EFh	┆
90h	4	B0h	ß	D0h	+	F0h	┆
91h	5	B1h	¸	D1h	┆	F1h	┆
92h	6	B2h	°	D2h	┆	F2h	┆
93h	7	B3h	©	D3h		F3h	┆
94h	8	B4h	®	D4h		F4h	┆
95h	9	B5h	™	D5h		F5h	┆
96h	0	B6h	'	D6h	┆	F6h	┆
97h	1	B7h	' 1	D7h	┆	F7h	┆
98h	2	B8h	' 2	D8h	=	F8h	┆
99h	3	B9h	' 3	D9h	≠	F9h	┆
9Ah	4	BAh	' 4	DAh	—	FAh	┆
9Bh	5	BBh	' 5	DBh	+	FBh	┆
9Ch	6	BCh	' 6	DCh	┆	FCh	┆
9Dh	7	BDh	' 7	DDh	┆	FDh	┆
9Eh	8	BEh	' 8	DEh	┆	FEh	┆
9Fh	9	BFh	' 9	DFh	┆	FFh	┆

Table B-3. Nationalized Video Display Characters

(Page 1 of 2)

Value	Canada	Finland/ Sweden	France/ Belgium	Germany	Italy	Latin Am./ Spain	Netherlands
A0	à	1/2	à	Ä	£	á	1/4
A1	â	Å	â	Ö	à	é	1/2
A2	ç	Ä	ä	Ü	è	í	3/4
A3	é	Ö	ç	ä	ì	ó	f
A4	è	à	é	ö	ò	ú	£
A5	ê	ä	è	ü	ù	û	IJ
A6	î	ö	ê	ß	1/3	ì	ij
A7	ô	ü	ë	à	1/4	ñ	à
A8	ù	É	î	è	1/2	ñ	é
A9	û	é	ÿ	ù	3/4	~	è
AA	ÿ	û	ô	é	°		ö
AB	ÿ	è	ö	â	◊	ø	ü
AC	ü	ô	ù	ê	ç	‰	â
AD	À	ü	û	î		ó	ê
AE	É		ü	ô	..	á	ù
AF	Û			û	é	°	ÿ
B0	î	⌘	Œ	Grave		ü	Umlaut
B1	ô	£	°	Acute	ÿ		Circumflex
B2	ç	Œ	£	Circumflex			°
B3	3/4	Acute	1/2	°			
B4	Â	Grave	Umlaut	µ		Umlaut	Grave
B5	Ê	Circumflex	Circumflex	Œ		Acute	Acute
B6							
B7							
B8							
B9	Û						
BA	É						
BB	Ë						
BC	ÿ						
BD							
BE	Û						
BF							

2486.B-3a

Table B-3. Nationalized Video Display Characters
(Page 2 of 2)

Value	Norway/ Denmark	Portugal	South Africa	Switzerland	Turkey	United Kingom	Yugoslavia
A0	Ü	Ŧ	^Circumflex	Ä	TL	1/4	£
A1	ü	đ	Umlaut	Ö	Ç	1/2	Ž
A2	Æ	à	Acute	Û	ç	3/4	ž
A3	æ	ã	Grave	ä	Ö	à	Đ
A4	Å	â	ê	ö	ö	â	đ
A5	ä	ç	ë	ü	ç	ä	š
A6	Ø	ç	ô	£	ç	ç	š
A7	ø	é	ö	à	Ş	é	Č
A8	Ö	e	à	e	ş	e	č
A9	ö	ê	é	ù	Ü	ê	Č
AA	Å	í	e	é	ü	î	č
AB	ä	ì	ü	â	ï	ô	1/4
AC	é	Õ	â	ê	i	ö	1/2
AD	á	ó	ù	î		ù	
AE	1/2	ò		ô		û	
AF	Grave	õ		û		ü	
B0	Umlaut	ô	£	Grave		£	
B1	£	ú	1/4	Acute			
B2	Š	ù	1/2	^Circumflex		Umlaut	
B3	Acute	£	3/4	°		Grave	
B4	à	Š	°	Umlaut		Acute	
B5	e		'n	Š		^Circumflex	

2466.B-3b

C

Keyboard Codes

Introduction to Appendix C

The keyboard codes generated by the K1 keyboard are shown in Table C-1 below. The keys for the K2, K3, K5, and SG101-K keyboards are essentially the same as those shown in Table C-1 except for the few differences noted. The differences are shown in Table C-2.

Table C-3 shows the keyboard codes generated by downstrokes on the SG102-K keyboard. Each upstroke on this keyboard generates the value F0h, followed by the key's downstroke value.

Legend For Keyboard-Code Tables

The key name is shown as it appears on the key cap label (for example, Finish, Shift).

Descriptions of the arrow keys are initially capitalized (for example, Left arrow).

In Table C-1, the four empty key posts covered by the double keys left-Shift, right-Shift, numeric-0, and Next are denoted by (SH-L'), (SH-R'), (0'), and (Next'), respectively.

The keys on the numeric pad are denoted "num 0", and so forth to distinguish them from the corresponding keys on the typewriter pad.

Table C-1. Keyboard Codes Generated by an Unencoded K1 Keyboard

(Page 1 of 4)

Keyboard Code (hexadecimal)	Key
00	Help
01	Up arrow
02	Mark
03	Bound
04	Finish
05	Prev Page
06*	1/2
07	Cancel
08	Backspace
09	Tab
0A	Return
0B	Down arrow
0C	Next Page
0D	Next
0E	Left arrow
0F	Right arrow
10*	(SH-L')
11*	Scroll Up
12	Move
13*	Scroll Down
14	Copy
15	f1
16	f2
17	f3
18	f4
19	f5
1A	f6

**Table C-1. Keyboard Codes Generated by an Unencoded K1
Keyboard**
(Page 2 of 4)

Keyboard Code (hexadecimal)	Key
1B	Go
1C	f7
1D	f8
1E	f9
1F	f10
20	Space
21	num 9
22*	(SH-R')
23*	(0')
24*	(Next')
25*	unused code
26*	unused code
27	' (single quote)
28*	unused code
29*	unused code
2A*	unused code
2B	-
2C	, (comma)
2D	- (hyphen)
2E	. (period)
2F	/
30...39	0...9
3A*	unused code
3B	;
3C*	unused code
3D*	unused code
3E*	unused code

**Table C-1. Keyboard Codes Generated by an Unencoded K1
Keyboard**
(Page 3 of 4)

Keyboard Code (hexadecimal)	Key
3F	invalid code
40	indicates the last key released; always has high bit on (that is, 0C0h)
41	num 6
42	num -
43	Action
44	Overtime
45	Lock
46	num 2
47	num 3
48	left Shift
49	right Shift
4A	num 0
4B	num .
4C	left Code
4D	right Code
4E..51	unused code
52*	unused code
53*	unused code
54*	unused code
55-58	unused code
59*	unused code
5B	[
5C	num 7
5D]
5E*	^ (caret)

**Table C-1. Keyboard Codes Generated by an Unencoded K1
Keyboard**

(Page 4 of 4)

Keyboard Code (hexadecimal)	Key
5F	unused code
60	num 1
61...7A	a...z
7B	num 4
7C	num 8
7D	num 5
7E	unused code
7F	Delete

* See Table C-2 for the K2, K3, K5, and SG101-K keys that generate this code.

Table C-2. Keyboard Codes: Key Differences for K2, K3, K5, and SG101-K Keyboards

Keyboard Code (hexadecimal)	K2, K3, K5, and SG101-K Keys
06	unused on all U.S. keyboards. Used on some International keyboards.
10	Jump
11	Scroll Next (SG101-K only) Scroll Up (K2, K3, and K5)
13	Scroll Prev (SG101-K only) Scroll Down (K2, K3, and K5)
22	Delete Char
23	unused code
24*	Num *
25*	Num/
26	Col
28	Page
29	Para
2A	Sent
3A*	Num =
3C	Word
3D	Line
3E*	Num %
52	System (K3, K5, and SG101-K)
53	Alt (K3, K5, and SG101-K)
54*	Num +
59*	Num 00
5E	unused on all U.S. keyboards. Used on some International keyboards.
5A	unused on all U.S. keyboards. Used on some International keyboards.

* For ReadActionCode and ReadActionKbd, these keys do not return the raw unencoded value that is returned in CTOS 3.3 and later versions of the operating system. Instead, the emulated value is returned.

**Table C-3. Keyboard Codes Generated by an Unencoded
SG102-K Keyboard**

(Page 1 of 5)

Keyboard Code (hexadecimal)	Key
00...06	unused code
07	f1
08	Escape
09...0C	unused code
0D	Tab
0E	,
0F	f2
10	unused code
11	left Control
12	left Shift
13	unused code
14	Caps Lock
15	q
16	1
17	f3
18	unused code
19	left Alt
1A	z
1B	s
1C	a
1D	w
1E	2
1F	f4
20	unused code
21	c

Table C-3. Keyboard Codes Generated by an Unencoded SG102-K Keyboard

(Page 2 of 5)

Keyboard Code (hexadecimal)	Key
22	x
23	d
24	e
25	4
26	3
27	f5
28	unused code
29	Space
2A	v
2B	f
2C	t
2D	r
2E	5
2F	f6
30	unused code
31	n
32	b
33	h
34	g
35	y
36	6
37	f7
38	unused code
39	right Alt
3A	m

**Table C-3. Keyboard Codes Generated by an Unencoded
SG102-K Keyboard**

(Page 3 of 5)

Keyboard Code (hexadecimal)	Key
3B	j
3C	u
3D	7
3E	8
3F	f8
40	unused code
41	<
42	k
43	i
44	o
45	0
46	9
47	f9
48	unused code
49	>
4A	/
4B	l
4C	;
4D	p
4E	- (hyphen)
4F	f10
50, 51	unused code
52	' (single quote)
53	unused code
54	[

**Table C-3. Keyboard Codes Generated by an Unencoded
SG102-K Keyboard**

(Page 4 of 5)

Keyboard Code (hexadecimal)	Key
55	-
56	f11
57	Print Screen
58	right Control
59	right Shift
5A	Enter
5B]
5C	\
5E	F12
5F	CTOS Lock
60	Down arrow
61	Left arrow
62	Pause
63	Up arrow
64	Delete
65	End
66	Backspace
67	Insert
68	unused code
69	num End
6A	Right arrow
6B	num Left arrow
6C	num Home
6D	Page Down
6E	Home

**Table C-3. Keyboard Codes Generated by an Unencoded
SG102-K Keyboard**

(Page 5 of 5)

Keyboard Code (hexadecimal)	Key
6F	Page Up
70	num Insert
71	num Delete
72	num down Arrow
73	num Scroll Lock
74	num right Arrow
75	num up Arrow
76	num Lock
77	num /
78	unused code
79	num Enter
7A	num Page Down
7B	unused code
7C	num +
7D	num Page Up
7E	num *
7F...83	unused code
84	num -

This page intentionally left blank.

Request Codes in Numeric Sequence

The Request codes listed here through 0BFFFh are reserved for future expansion; they should not be used by system builders. Request codes 0C000h-0FFFFh are available for system builder use.

Request Code	Operation Name
-16 or 65520	TerminateMcr*
-12 or 65524	GetKbdId
00	(illegal)
01	SetPath
02	ClearPath
03	SetPrefix
04	OpenFile
05	CreateFile
06	DeleteFile
07	RenameFile
08	GetFileStatus
09	SetFileStatus
10	CloseFile
11	MountVolume
12	DismountVolume
13	ChangeFileLength
14	GetDateTime
15	GetVhb
16	SetDevParams
17	CreateDir
18	DeleteDir
19	CloseAllFiles
20	QuietIO (internal)
21	QueryVidHdw
22	LoadFontRam
23	LoadSyleRam
24	LoadCursorRam
25	ReadDirSector
26	(reserved)
27	GetUcb

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

Request Code	Operation Name
28	Chain
29	LoadTask
30	SetFhLongevity
31	GetFhLongevity
32	ResetSubsys (internal)
33	(reserved)
34	(reserved)
35	Read
36	Write
37	ReadIdAndData (internal)
38	Format
39	DeviceReadId (internal)
40	AllocExch
41	DeallocExch
42	AllocMemorySL
43	DeallocMemorySL
44	AllocMemoryLL
45	DeallocMemoryLL
46	AllocAllMemorySL
47	ResetMemoryLL
48	QueryMemAvail
49	OpenRtClock
50	CloseRtClock
51	SetDateTime
52	Beep
53	ReadKbd
54	ReadKbdDirect
55	QueryKbdLeds
56	SetKbdLed
57	SetKbdUnencodedMode
58	QueryKbdState
59	SetSysInMode
60	ReadActionCode
61	QueryWsNum
62	CloseAllFilesLL
63	KbdWakeUp (internal)
64	BeeperOff (internal)
65	SetKbdUnencodedModeReal (internal)
66	KbdResetSysIn (internal)
67	DisableActionFinish
68	CheckpointSysIn
69	SetIntHandler
70	ResetKbd (internal)*
71	ResetSysIn (internal)*
72	ResetAgent (internal)*
73	(reserved)
74	ResetVideo
75	InitVidFrame
76	InitCharMap
77	SetScreenVidAttr
78	CloseISAM
79	CreateISAM
80	DeleteISAM
81	DeleteISAMRecord

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

Request Code	Operation Name
82	GetISAMRecords
83	LockISAM
84	ModifyISAMRecord
85	OpenISAM
86	ReadISAMRecordByUri
87	ReadNextISAMRecord
88	ReadUniqueISAMRecord
89	RenameISAM
90	SetISAMProtection
91	SetupISAMIterationKey
92	SetupISAMIterationPrefix
93	SetupISAMIterationRange
94	StoreISAMRecord
95	UnlockISAM
96	PurgeISAMUser (internal)*
97	OpenFileLL
98	ConvertToSys
99	ServeRq
100	GetClusterStatus
101	SetCommISR
102	ResetCommISR
103	KbAttn3270 (internal)
104	ScreenRead3270 (internal)
105	StatusRead3270 (internal)
106	ReadyForCmd3270 (internal)
107	StartEm3270 (internal)
108	StopEm3270 (internal)*
109	CancelRq3270 (internal)
110	ReportStatus3270 (internal)
112	FileSystemAbort**
113	QueryFrameAttrs
114	QueryFrameString
115	(reserved)
116	(reserved)
117	(reserved)
118	(reserved)
119	(reserved)
120	(reserved)
121	SetLPlsr
122	DisableCluster
123	GetRunFileHdr (internal)
124	QueryDcb
125	WriteLog
126	SetCommISRRaw (internal)
127	PurgeISAMTransaction
128	EndISAMTransaction
129	GetISAMRecordsHold
130	HoldISAMRecord
131	ReadISAMRecordByUriHold
132	ReadNextISAMRecordHold

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

** This is an abort Request. See the AbortRequests page in "Operations" for its format.

Request Code	Operation Name
133	ReadUniqueISAMRecordHold
134	ReleaseISAMRecord
135	SetupISAMIteration
136	StartISAMTransaction
137	AddQueueEntry
138	RemoveKeyedQueueEntry
139	ReadNextQueueEntry
140	ReadKeyedQueueEntry
141	MarkNextQueueEntry
142	MarkKeyedQueueEntry
143	RemoveMarkedQueueEntry
144	UnmarkQueueEntry
145	RewriteMarkedQueueEntry
146	EstablishQueueServer
147	TerminateQueueServer
148	PurgeQueueServer (internal)**
149	SignOffRJE
150	SignOnRJE
151	StatusRJE
152	AcceptCommCall
153	CloseAllCommLines*
154	CloseCommLine
155	DialComm
156	DisconnectComm
157	FlushCommBuffer
158	GetCommParameters
159	OpenCommLine
160	ReadComm
161	SetCommParameters
162	WriteComm
163	BreakComm
164	ResetQueMgr*
165	NotifyNextIncomingCall
166	AcceptX25Call
167	InitiateX25Call
168	ClearX25Call
169	PurgeX25User*
170	ReadX25Packet
171	WriteX25Packet
172	WriteX25Interrupt
173	ResetX25Call
174	QueryX25Status
175	ConnectX25Permanent
176	RemovePartition
177	GetPartitionHandle
178	LoadPrimaryTask
179	TerminatePartitionTasks
180	VacatePartition
181	CreatePartition
182	SetPartitionLock
183	SetPartitionExchange
184	GetPartitionExchange
185	GetPartitionStatus

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

** This is an abort Request. See the AbortRequests page in "Operations" for its format.

Request Code**Operation Name**

186	SetExitRunFile
187	QueryExitRunFile
188	ConfigureSpooler
189	SpoolerPassword
195	PurgeTapeUser*
197	ResetSplr (internal)*
198	ModifyISAMRecordByKey
199	DeleteISAMRecordByKey
200	LogRemote (internal)
201	VacateParCleanup (internal)
202	GetWsUserName
203	SetWsUserName
215	ChangeOpenMode
216	GetDirStatus
217	SetDirStatus
218	ChangeUserNumQmg++
219	ChangeUserNumFs++
229	PurgeSNAUser**
233	TerminateSNAUser*
239	PurgeMailUser*
251	PurgeMailUniqueId**
252	SetNode
253	ExpandSpec
255	SetVideoTimeout
256	ReadActionKbd
257	ReadKbdStatus
259	ChangeUserNumMail++
260	QuietMailUser+
261	OsVersion
262	LoadInteractiveTask
264	MapXBusWindow
265	QuietKbdForSwap+
267	PurgeX25**
268	SetTrapHandler
270	CtNetTermination*
271	QueryRequestInfo
272	GetUserStatus
273	CMTermination*
274	FilterProsTermination*+
278	AllocaAreaSL
279	ExpandAreaSL
280	ExpandAreaLL
281	ShrinkAreaSL
282	ShrinkAreaLL
283	QueryBigMemAvail
284	CreateAlias
285	SetSegmentAccess

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

** This is an abort Request. See the AbortRequests page in "Operations" for its format.

+ This is a swapping Request. See the SwappingRequests page in "Operations" for its format.

++ This is a change user number Request. See the ChangeUserNumberRequests page in "Operations" for its format.

Request Code	Operation Name
286	SetColorData (internal)
290	ResetMemorySL
291	ResetAllSegs
292	FreeLargeLL
293	LoadFile
294	ReopenFile
295	SwapInContext
296	CreateUser
297	GetHandleStatus
302	GetNodeName
303	ReadSwap
304	WriteSwap
305	DeactivateRunFile
306	GetMemoryInfo
308	AllocUserNumbers
309	DeallocUserNumbers
310	SetDeviceHandler
311	RemoteBootCopy (internal use)
312	GetRouteTable (internal use)
313	UpdateRouteTable (internal use)
314	QueryBoardInfo
318	DoDirectRead (internal use)
319	DoDirectWrite (internal use)
320	DefineLocalPageMap
321	MapXBusWindowLarge
322	Set386TrapHandler
323	ServiceOverlayA
324	ServiceOverlayB
325	AllocMemoryFramesSL
326	InitLocalPageMap
327	LinkFile
328	MakeDirectory
329	RemoveDirectory
331	CreateBigPartition
332	SgFromSa
333	AllocLdtSlot
334	FsCableDrop
335	FilterDebugInterrupts
336	RemakeAliasForServer
337	DeallocAliasForServer
338	AllocMemoryPermanent
339	SetDefaultTrapHandler
340	QueryDeviceNames
341	ShrinkPartition
344	InstallNet
345	DeallocSg
346	QueryDeviceName
347	ReadKbdDataDirect
348	SetIBusHandler
349	ResetIBusHandler
350	LoadRunFile
351	DeallocRunFile
352	SetFSConfigParams
353	GetScsiInfo
353	ScsiQueryInfo (new alias for GetScsiInfo)
354	ReservePartitionMemory
355	ScsiOpenPath

Request Code	Operation Name
356	ScsiQueryPathParameters
357	ScsiSetPathParameters
358	ScsiClosePath
359	ScsiReset
360	ScsiCdbDataIn
361	ScsiCdbDataOut
362	ScsiRequestSense
363	ScsiManagerNameQuery
364	GetPartitionSwapMode
373	LockInCache
374	UnlockInCache
387	ScsiTargetDataReceive
388	ScsiTargetDataTransmit
389	ScsiTargetCdbCheck
390	ScsiTargetCdbWait
391	SerialNumberQuery
392	ConfigurationQuery
395	LockFile
396	UnlockFile
397	GetVerifyCode
398	SetVerifyCode (internal)
399	ReadOsKbdTable
400	PostKbdTable
402	WriteKbdBuffer
403	ReadKbdInfo
413	NameTermination*
436	DmaMapBuffer
448	SetKeyboardOptions
449	GetKeyboardId
461	ResetTrapHandler
464	SetIOOwner
466	QueryIOOwner
467	MapPhysicalAddress
468	UnmapPhysicalAddress
469	ResetDeviceHandler
470	QueryTrapHandler
821	AccessVersion
4096	ChangeUserNumISAM++
4097	QuietISAMUser+
4099	QuietQMGR+
4100	QuietSplr+
4101	ExecFh
4102	InitCommLine
4103	ResetCommLine
4104	ChangeCommLineBaudRate
4105	TerminateCommLine*
4106	ChangeUserNumComm++
4110	RemakeFh
4118	TerminateVideoClient (internal)*

-
- * This is a termination Request. See the TerminationRequests page in "Operations" for its format.
 - + This is a swapping Request. See the SwappingRequests page in "Operations" for its format.
 - ++ This is a change user number Request. See the ChangeUserNumberRequests page in "Operations" for its format.

Request Code	Operation Name
4121	GoDebugger
4130	TimerRequestBlock
4155	SignOnLog
4193	McrFillBuf
8192	ReadHardId
8202	LockXbis
8203	UnlockXbis
8204	SetXBusMIsr
8205	ResetXBusMIsr
8208	RkvsVersion
8210	QueueMgrVersion
8211	SpoolerVersion
8212	MouseVersion
8216	TsVersion
8217	XbifVersion
8224	CfaServerVersion
8225	CfaWaVersion
8226	CfaFFVersion
8227	DcxVersion
8228	McrVersion
8230	StatisticsVersion
8234	ReadMcr
8235	DeinstallMcr (internal use)
8236	PurgeMcr
12290	RemoteBoot
12304	UpdateFpMountTable
12306	OpenTerminal
12307	CloseTerminal
12308	WhereTerminalBuffer
12309	ReadTerminal
12310	SetTerminal
12313	DrainTerminalOutput
12324	UpdateDcb 12325 ReadRemote
12326	WriteRemote
12351	TapeRegisterSlave
12352	LocalOpenTape
12353	LocalPurgeTapeUser*
12356	LocalConfigureSpooler
12357	LocalSpoolerPassword
12358	LocalResetSplr*
12360	MegaframeDisableCluster
16393	QueryNodeName

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

Request Code	Operation Name
20482	CtNetAbort**
32797	LockXbis
32798	UnlockXbis
32799	SetXBusMIsr
32800	ResetXBusMIsr
32801	TsVoicePlaybackFromFile
32802	TsVoiceRecordToFile
32803	TsVoiceConnect
32804	TsVoiceStop
32805	TsDataOpenLine
32806	TsDataRead
32807	TsDataWrite
32808	TsDataCheckpoint
32809	TsDataCloseLine
32810	TsDataChangeParams
32811	TsDataRetreiveParams
32812	TsDataUnacceptCall
32813	TsGetStatus
32814	TsConnect
32815	TsDial
32816	TsReadTouchTone
32817	TsDoFunction
32818	TsOffHook
32819	TsOnHook
32820	TsHold
32821	TsQueryConfigParams
32822	TsSetConfigParams
32823	TsRing
32824	TsDeinstall
32865	AddQueue
32866	CleanQueue
32867	DeinstallQueueManager
32868	GetQMStatus
32869	RemoveQueue
33119	RescheduleMarkedQueueEntry
33137	ResetVideoGraphics
33139	CdAbortUser (internal)**
33140	CdTerminateUser (internal)*

* This is a termination Request. See the TerminationRequests page in "Operations" for its format.

** This is an abort Request. See the AbortRequests page in "Operations" for its format.

Request Code	Operation Name
33141	CdSwapUser (internal)+
33142	CdChangeUserNumber (internal)++
33145	CdServiceControl
33171	AsGetVolume
33174	AsSetVolume
33201	TsLoadCallProgressTones
33203	CdAbsoluteRead
33205	CdAudioCtl
33206	CdControl
33207	CdGetDirEntry
33208	CdGetVolumeInfo
33209	CdRead
33210	CdVersion
33212	CdOpen
33213	CdClose
33214	CdSearchClose
33215	CdSearchFirst
33216	CdSearchNext
33217	CdDirectoryList
33218	CdVerifyPath
33225	SeqAccessVersion
33226	SeqAccessOpen
33227	SeqAccessClose
33228	SeqAccessModeQuery
33229	SeqAccessModeSet
33230	SeqAccessControl
33231	SeqAccessStatus
33232	SeqAccessWrite
33233	SeqAccessRead
33234	SeqAccessCheckpoint
33235	SeqAccessRecoverBufferData
33236	SeqAccessDiscardBufferData
36932	SeqAccessTermination (internal)*
49232	SetKbdSemicodedMode
49234	ReadKbdSemiCoded
65520 (or -16)	TerminateMcr*
65521 (or -15)	McrFillBuf (internal use)
65524 (or -12)	GetKbdId

-
- * This is a termination Request. See the TerminationRequests page in "Operations" for its format.
 - + This is a swapping Request. See the SwappingRequests page in "Operations" for its format.
 - ++ This is a change user number Request. See the ChangeUserNumberRequests page in "Operations" for its format.

The currently defined formats and default templates, along with a description, examples, and maximum length, are listed in Table E-1. The maximum lengths given refer to the length of the actual date/time string produced by `NlsFormatDateTime`, not the format string.

NOTE: When changing the templates and the various strings used to construct the dates, do not exceed the maximum string length. Allow for the maximum length when using one of the defined formats.

The following rules summarize how `NlsFormatDateTime` interprets the templates:

1. Text contained within exclamation marks (!) gets expanded according to the date and time as passed to `NlsStdFormatDateTime`. Other text is reproduced verbatim.
2. Letters expand to attributes of the specified date and time, as follows:

Letter	Date and Time
a:	am, pm, noon, or midnight
d:	day of the month (numeric)
g:	Emperor year (years since start of Emperors' reign)
h:	hour (numeric, 12 hour clock)
m:	minute (numeric)
n:	month of year (alphabetic)
o:	month of year (numeric)
s:	second (numeric)
t:	hour (numeric, 24-hour clock)
w:	day of week (alphabetic)
y:	year (numeric)

Table E-1. NLS Templates

Description	Identifier	Default Templates	Examples	Maximum Length
Columnar numeric date and time of day	0	l0ol/l0dl/l0yl lhhl:l0ml l2*A	7/18/85 9:03AM 10/08/85 11:13PM	16
Numeric date and time of day	1	l*ol/l0dl/l0yl l*hl:l0ml l2*A	7/18/85 9:03AM 10/08/85 11:13PM	16
Columnar date and time of day	2	lNnnl lddl lyyyyl lhhl:l0ml l2*A	Jul 1, 1985 9:03 AM Jul 18, 1985 11:13 PM	26
Date and time of day	3	lNnnl l*d, lyyyyl l*hl:l0ml l2*A	Jul 1, 1985 9:03AM Jul 18, 1985 11:13PM	26
Columnar day, date, and time of day	4	lWwwl lNnnl lddl, lyyyyl lhhl:l0ml l2*A	Mon Jul 1, 1985 9:03 AM Mon Jul 18, 1985 11:13 PM	30
Day, date, and time of day	5	lWwwl lNnnl l*d, lyyyyl l*hl:l0ml l2*A	Mon Jul 1, 1985 9:03 AM Mon Jul 18, 1985 11:13 PM	30
Columnar 24-hour time of day†	6	l0tl:l0ml	23:43 07:00	5
24-hour time of day†	7	l*tl:l0ml	23:43 7:00	5
Columnar time of day	8	lhhl:l0ml l2*A 7:00AM	11:43PM	8
Time of day	9	l*hl:l0ml l2*A	11:43PM 7:00AM	8
Columnar numeric date	10	l0ol/l0dl/l0yl	11/18/85 01/01/01	8
Numeric date	11	l*ol/l*d/l0yl	11/18/85 1/1/01	8
Time	12	l0hl:l0ml:l0sl	09:03:05	8
Temporary file name	13	l0tl:l0ml:l0sl.tmp 23:03:05.tmp	09:03:05.tmp	30
Long day and date	14	lW*wl lN*nl l*d, l*yl	Monday June 3, 1985	45
Long date	15	lN*nl l*d, l*yl	June 3, 1985	30
Columnar date/time for file names‡	16	l0ol/l0dl/l0yl-l0hl:l0ml:l0sl l2*A	01/01/01-07:09:05AM	40
Abbreviated date	17	lNnnl l*d lYYYY	Jun 3, 1985	30

†For applications that want 24-hour time regardless of country.

‡Various strings are normally appended to construct file names. No spaces should be allowed.

The actual letters may be selected via the NLS data tables. This provides the ability for application programs to accept template specifications from users in a natural fashion based on country.

3. The attributes expand to as many character positions as are held by adjacent identical letters. For example, lyyyy! expands to "1985" and !yy! expands to "85". Left truncation is used for numeric fields; right truncation is used for alphabetic fields. By default, template fields that are longer than the attribute filled in are padded with spaces. Numbers are padded to the left; strings to the right.
4. Template letters that are capitalized result in capitalization of the corresponding letters in the expanded string. For example, !AA! expands to "AM," !aa! expands to "am," and !Aa! expands to "Am."
5. An asterisk (*) before a template letter defines a variable format repetition. The asterisk causes the next format character to be used repetitively until the entire expansion is complete. For example, !*d! results in "1" or "12". Also, !*n! results in "september" while !*N! results in "SEPTEMBER".
6. An asterisk may be preceded by a series of like format characters (case may differ). !N*n! is an example. This is useful in controlling the case of the leading character in variable length fields. The example, !N*n!, would result in "September".
7. Zeroes before a template letter result in leading zeroes preceding the significant part of the number. For example, !0d! results in "09" or "12".
8. The first numeric digit (and first character) within a template field, such as the digit "2" in !2*w!, causes the selection of a date name table. In this case, date name table number 2 is selected. This selection remains in effect until a new table is selected or the date/name string is complete.

This type of template is used, for example, to select alternate spellings for the case where the abbreviation for a month or day name is not merely the left-most letters of the full name. A leading 0 is not a table identifier. A template with a leading 0 that is to result in a string is illegal.

Templates are processed left to right. The first table (number 1) is assumed at the start of processing for a template. If a change of table number is found within a template, it takes effect immediately and remains in effect until changed or until the end of the template is reached. Appearance of a numeric digit within a template field has no effect on formatting. If a template calls for a date name table that does not exist, the message, "Invalid template index," is returned to the user.

A small set of options is standardized to handle the varying requirements of application programs for different types of date and time formats. These date and time formats are selected by an ID code. ID codes 0 through 32767 are reserved; 32768 through 65535 are available for definition.

Message File Macro Definitions

To allow added flexibility in outputting messages, the messages in a message file can contain macros. Message file macros serve as place holders in a message for data that is to be inserted into the message at run time.

The macros may be expanded with data supplied by using message file operations such as `ExpandLocalMsg`, `GetMsg`, `GetAltMsg`, `PrintMsg`, or `PrintAltMsg` or by programs calling such operations.

Macros Not Under Program Control

Macro	Meaning
%U	Insert the workstation user name in the message. Note that %U expands to NULL if the user name in the Application System Control Block is of 0 length.
%Dn.	Insert the system date/time in the message formatted with template number n.

See Appendix E for the date/time template formats, descriptions, and examples. The templates are defined in the file, `NlsDateTables.asm`.

%Kn.	Insert the keycap text string for key cap index number n into the message.
-------------	--

See the description of `GetNlsKeycapText` in Chapter 3, "Operations." It contains the key cap index numbers and their corresponding text strings. Keycap text is defined in the file, `KeycapText.asm`.

Macros Under Program Control

Macro

Meaning

%nT

Insert the program-supplied parameter number *n*.
T is a string denoting format options for the number *n*.

T can be any of the following values:

Value

Meaning

Dm date/time (where *m* is the
template ID code)

N unsigned decimal-based
integer

H unsigned hexadecimal-based
integer (See format options below.)

S string

If *T* is not specified, *S* is the default.

The macro %6D4, for example, means the sixth array element will be expanded as a date using template ID code 4 in the date/time templates.

See the descriptions of ExpandLocalMsg, GetMsg, GetAltMsg, PrintMsg and PrintAltMsg in Chapter 3, "Operations." *pRgSd* and *sRgSd* describe the array that you include in your program for supplying the parameters to be inserted into messages.

Format Options for Macro Option H

The hexadecimal macro option (H) expands the string to a fixed number of places based on the count of bytes (1, 2, or 4) in the string descriptor array element. As an example, a value of 100 can return one of three different strings, depending on the string size as follows:

Value	String Size (in bytes)	Returned String
100	1	64
100	2	0064
100	4	00000064

For other format options, use OutputWord or OutputQuad described in "Operations."

X-BUS and I-Bus Module IDs

This appendix lists the module identification numbers (IDs) of all the devices that can be connected to an X-Bus or I-Bus. A *module ID* is a word value consisting of a *module type code* (high byte) and a *device attribute code* (low byte). The module type code uniquely identifies the name of an X-Bus or I-Bus device such as a floppy disk or keyboard. The bit settings in the device attribute code provide additional information about the device to the processor boot ROM.

Bit Meanings in a Module ID

The module ID consists of two bytes: the module type code is contained in the high byte and the attribute code, in the low byte. The bit format of the ID is shown below:

Module ID

Type Code	Attribute Code
<i>tttt tttt</i> 15 8	<i>gbdd xxvv</i> 7 0

The bits have the following meanings:

- | | |
|------------------|---|
| <i>tttt tttt</i> | is a constant identifying the module type. For storage modules, the low 4 bits are described in detail in "Storage Module Type Codes," later in this appendix. |
| <i>g</i> | indicates the module is a video controller with its own ROM. (<i>g</i> is 0 for an I-Bus module.) The contents of this ROM can be read over the DMA channel specified. |

If no DMA channel is specified (the bits *dd* are 0), the ROM can be read into memory by opening an X-Bus window. (The bits *dd* are usually 0 for an I-Bus module.) All device ROMs are byte oriented; thus one byte of ROM is read into each word of memory; then the bytes are packed.

b indicates the module is a bootable device with its own boot ROM. (*b* is 0 for an I-Bus module.) When this bit is set in the device control register, the contents of this ROM can be read into main memory using the specified DMA channel, and then used to boot the processor from the X-Bus module. If no DMA channel is specified (the bits *dd* are 0), the ROM can be read into memory by opening an X-Bus window. (The bits *dd* are usually 0 for an I-Bus module.) All device ROMs are byte oriented; thus one byte of ROM is read into each word of memory; then the bytes are packed.

dd specifies which DMA channel to use when reading a video controller ROM or boot ROM. (*g* or *b* must be set.) When these bits are 0, an X-Bus window must be used to read the contents of the ROM into memory.

xx reserved for expansion; must be 0's.

vv indicates the module version number. This number qualifies the module ID so enhanced versions can be introduced without changing the module type code. If *b* is 1 and *vv* is 0, the module will only boot on an 80186-based CPU. If *vv* > 1, the module will boot on 80286- and 80286-based CPUs as well.

Obtaining Module IDs

The GetModuleId operation returns the module ID of a specified X-Bus or I-Bus device. (See the description of this operation in Chapter 3.) If you are going to compare the word value returned to the value of a device, for example

If (ModuleIDRet == 10)

.
.
.

be sure to mask the bits in the attribute code before making the comparison.

Storage Module Type Codes

Module type codes in the range 10h through 1Fh and 20h through 2Eh identify disk modules. The bit format of the module type code is shown below:

0001 (or 0010) *xewu*

The 4 low bits have the following meanings:

- | | |
|----------|---|
| <i>x</i> | is reserved for future use (currently 0). |
| <i>e</i> | is the expansion bit. If <i>e</i> =1, a hard disk expansion is present. <i>e</i> is 0 on modules with type codes in the range 20h through 2Eh. |
| <i>w</i> | is the expansion width bit. If <i>w</i> =1, the hard disk expansion is full width; otherwise, the expansion is half width. <i>w</i> is 0 on modules with type codes in the range 20h through 2Eh. |
| <i>u</i> | is the upgrade bit. If <i>u</i> =1, the module is a hard disk upgrade and contains no floppy disk; otherwise, the module contains a hard disk and a floppy. |

The disk modules do not encode the size or characteristics (such as the number of cylinders) of a disk.

Modules with type codes 10h through 1Fh have hard disks that conform to the ST-506 standard and (optionally) floppy disks controlled by the Western Digital WD-2797 controller.

Modules with type codes 20h through 2Eh have a SCSI device (usually a hard disk) and (optionally) a floppy disk controlled by a NEC-765 compatible controller.

Module IDs

Table G-1 lists the module IDs. In the table, X indicates whether the device is an X-Bus or an I-Bus device. Module type codes in the range 80h through FFh that are not reserved or already assigned are available for customer use. To register a module ID, contact Unisys Distributed Systems Division.

NOTE: This table lists the module IDs for Convergent and Unisys modules known at the date of publication. It does not specify hardware equivalencies. If you have questions, please contact your Unisys Technical Support representative.

Table G-1. Module IDs

(Page 1 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
00h			X	Dual floppy disk (B24-M1, B25-MF1, FD-001, FD-0A1)
01h	00h		X	8 Mhz 80186 CPU (CP-001, B26)
02h	00h	X		Any monitor
03h				Reserved
04h		X		K1 keyboard (KM-001, K1)
05h-06h				Reserved
07h			X	8-Mhz 80286 CPU (CP-002, CP-0A2, B28)
08h		X		200 dots/inch mouse (PD-001, B25-MO3, SG-100-U, SG-101-U)
09h			X	8-Mhz 80186 CPU (CWS)
0Ah			X	16-Mhz 80386 CPU (CP-003, CP-0A3, B38), 25-Mhz 80386 CPU (B38-EXP)
0Bh			X	Series 286i CPU
0Ch		X		100 dots/inch mouse
0Dh	00h			Null module
0Fh	00h		X	Series 386i, B39
10h	71h		X	Hard disk/floppy disk 96 tpi

Table G-1. Module IDs

(Page 2 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
11h	71h		X	Hard disk upgrade
12h-13h			X	Reserved
14h	71h		X	Hard disk/floppy disk 96 tpi with half-width expansion
15h	71h		X	Hard disk upgrade with half-width expansion
16h	71h		X	Hard disk/floppy disk with full-width expansion
17h	71h		X	Hard disk upgrade with full-width expansion
18h			X	Unisys special disk product
19h			X	Unisys special disk product
1Ah-1Fh				Reserved
20h	41h		X	SCSI hard disk/floppy disk
21h	41h		X	SCSI hard disk upgrade
21h	B0h	X		SG101-K keyboard (USA)
22h	B0h	X		SG101-K keyboard (Arabic)
23h	B0h	X		SG101-K keyboard (Belgium)
24h	B0h	X		SG101-K keyboard (Canada)
25h	B0h	X		SG101-K keyboard (Canada)

Table G-1. Module IDs

(Page 3 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
26h	B0h	X		SG101-K keyboard (French)
27h	B0h	X		SG101-K keyboard (German)
28h	B0h	X		SG101-K keyboard (New Greek)
29	B0h	X		SG101-K keyboard (Hebrew)
2A	B0h	X		SG101-K keyboard (Iceland)
2B	B0h	X		SG101-K keyboard (Italy)
2C	B0h	X		SG101-K keyboard (Netherlands)
2D	B0h	X		SG101-K keyboard (Norway/Denmark)
2E	B0h	X		SG101-K keyboard (Portugal)
2Fh			X	PC-Emulator
2Fh	B0h	X		SG101-K keyboard (South Africa)
30h	B0h	X		SG101-K keyboard (Spain/Latin America)
31h	41h		X	HB-001 Tape Backup
31h	B0h	X		SG101-K keyboard (Swiss-German)
32h	00h		X	Telephone Manager
32h	B0h	X		SG101-K keyboard (Swiss-French)
33h	00h		X	Telephone Manager (spare)
33h	B0h	X		SG101-K keyboard (Turkey)
34h	00h		X	Ethernet controller (XE-001)

Table G-1. Module IDs

(Page 4 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
34h	B0h	X		SG101-K keyboard (United Kingdom)
35h	00h		X	Ethernet controller (spare)
35h	B0h	X		SG101-K keyboard (Yugoslavia)
36h	00h		X	Universal communications processor
37h	00h		X	XC-002 Multiline port expander
38h	00h		X	Graphics controller (GC-001, B25-GRA) with color monitor
39h	00h		X	Graphics controller (GC-001) with monochrome monitor or Graphics controller (B25-GPP) with monitor (B25-PD7 or B25-PD8)
3Ch	80h		X	GC-003 with VM-003 connected
3Dh				Reserved for video
3Eh	80h		X	GC-003 with VM-001/VM-002
3Fh	80h		X	GC-003 with VC-002 connected
40h	00h		X	GC-001
41h	80h		X	GC-002
42h			X	Reserved for Multislot X-Bus module

Table G-1. Module IDs

(Page 5 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
43h			X	Reserved for Communications module
44h	80h		X	GC-102
45h	00h		X	Reserved for Voice Processor
46h	00h		X	Reserved for Voice Processor
47h				Reserved
48h-4Fh				Reserved
50h	80h		X	Graphics controller (GC-X04, B25-VGX) with monitor (B25-GS1, VM-005)
51h	80h		X	Reserved for graphics controller (GC-X04, B25-VGX)
52h	80h		X	Graphics controller (GC-X04, B25-VGX) with Unisys positive polarity monitor (B25-PD7, B25-PD8, VM-004)
53h	80h		X	Reserved for graphics controller (GC-X04, B25-VGX)
54h	80h		X	Graphics controller (GC-X04, B25-VGX) with VM-003 high resolution monochrome monitor
55h	80h		X	Graphics controller (GC-X04, B25-VGX) with no monitor
56h	80h		X	Graphics controller (GC-X04, B25-VGX) with monitor (VM-001, VM-002, B25-D1, B25-D2)

Table G-1. Module IDs

(Page 6 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
57h	80h		X	Graphics controller (GC-X04, B25-VGX) with monitor (VC-002, B25CD3)
58h	80h		X	Graphics controller (GC-X04, B25-VGX) with monitor (8514 compatible, B25-CA1, VC-003)
59h	80h		X	Graphics controller (GC-X04, B25-VGX) with monitor (8513 compatible, 8503 compatible, B25-VA1), or with monitor package (B25-VDC, B25-VKA)
5Ah-5Fh			X	Reserved for GC-X04
60h-63h				Reserved for future CPU
64h				Reserved
65h	B0h	X		SG102-K keyboard
66h-80h				Reserved
81h		X	X	Assigned to an OEM
82h		X	X	Assigned to an OEM customer
83h			X	Assigned to an OEM customer
84h	00h		X	Tape streamer (B25-TS1)
85h-87h				Reserved
88h			X	Assigned to an OEM customer
89h	70h		X	40Mb and 80Mb Winchester drives (ECC capable)

Table G-1. Module IDs

(Page 7 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
8Ah	70h		X	Floppy + Winchester drives (ECC capable)
8Bh-8Eh				Reserved for future disk products
8Fh	80h		X	Advanced Graphics Processor
8Fh	81h		X	Advanced Graphics Processor
90h		X		OEM special K4 keyboard
91h		X		Assigned to an OEM customer
92h		X		Assigned to an OEM customer
93h		X		Assigned to an OEM customer
94h		X		Assigned to Unisys special product
95h			X	Assigned to Unisys special product
96h			X	Assigned to Unisys special product
97h			X	Assigned to Unisys special product
98h				Assigned to an OEM
99h-9Ah				Unassigned
9Bh			X	Assigned to Unisys special product
9Ch-A8h				Unassigned
A9h	70h		X	40Mb and 80Mb Winchester drives
AAh			X	Reserved for disk products
ABh			X	Assigned to Unisys special product

Table G-1. Module IDs

(Page 8 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
ACh-AFh				Unassigned
B0h			X	Assigned to an OEM
B1h		X		Assigned to an OEM
B2h			X	Assigned to an OEM
B3h			X	Assigned to an OEM
B4h-B6h				Unassigned
B7h			X	Four-port communications I/O module (B25-DCX)
B8h			X	Reserved
B9h-BCh				Unassigned
BDh			X	Assigned to Unisys special product
BEh				Unassigned
BFh	00h		X	Assigned to Unisys special product
C0h			X	Assigned to Unisys special product
C1h-C2h				Unassigned
C3h	01h		X	IEEE 802.3 LAN, Ethernet LAN Module (B25-EN3)
C4h	00h		X	LAN Token Ring Module (B25-TR2)
C5h	01h		X	Universal communications processor (IDS), Intelligent Data Communication Module

Table G-1. Module IDs

(Page 9 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
C5h	02h		X	Universal communications processor (IDS) (X.21 1984 standard) B25-ID2
C6h			X	Network attachment module
C7h	00h		X	Reserved for future IDS
C8h			X	ISDN Module (B25-DN1, B25-DN2)
C9h				Reserved for intelligent communications modules
CAh		X		K2 French keyboard (Canada)
CBh		X		K2 new keyboard (Canada)
CCh-CDh				Reserved for intelligent modules
CEh-CFh		X		I-Bus synchronization byte. Do not use.
D1h		X		K2 keyboard (Portugal)
D2h		X		K2 keyboard (Yugoslavia)
D3h		X		SST terminal keyboard
D4h-D6h		X		Reserved for keyboards
D7h		X		K3, K5 keyboards (Iceland)
D8h		X		K3, K5 keyboards (Hebrew)
D9h		X		K3, K5 keyboards (Arabic)
DAh		X		K3, K5 keyboards (Greek)
DBh		X		K3, K5 keyboards (Portugal)

Table G-1. Module IDs

(Page 10 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
DCh		X		K3, K5 keyboards (Yugoslavia)
DDh		X		K3, K5 keyboards (Norway/Denmark)
DEh		X		I-Bus mouse (B27)
DFh		X		I-Bus synchronization byte. Do not use.
E0h		X		K3, K5 keyboards (Turkey)
E1h		X		K3, K5 keyboards (Belgium)
E2h		X		K3, K5 keyboards (United Kingdom)
E3h		X		K3, K5 keyboards (Swiss/German)
E4h		X		K3, K5 keyboards (Swiss/France)
E5h		X		K3, K5 keyboards (South Africa)
E6h		X		K3, K5 keyboards (Netherlands)
E7h		X		K3, K5 keyboards (Spain/Latin America)
E8h		X		K3, K5 keyboards (Italy)
E9h		X		ALC keyboard CER I-311
EAh		X		K3, K5 keyboards (Germany)
EBh		X		K2 keyboard (Arabic)
ECh		X		K3, K5 keyboards (France)
EDh		X		K3, K5 keyboards (Finland/Sweden)
EEh		X		K2 keyboard (Turkey)

Table G-1. Module IDs

(Page 11 of 11)

Type Code	Attribute Code	I-Bus	X-Bus	Device Type
EFh		X		K3, K5 keyboards (USA)
F0h		X		K2 keyboard (Belgium)
F1h		X		K2 keyboard (United Kingdom)
F2h		X		K2 keyboard (Swiss/German)
F3h		X		K2 keyboard (Swiss/France)
F4h		X		K2 keyboard (South Africa)
F5h		X		K2 keyboard (Norway/Denmark)
F6h		X		K2 keyboard (Netherlands)
F7h		X		K2 keyboard (Spain/Latin America)
F8h		X		K2 keyboard (Italy)
F9h		X		K2 keyboard (Germany)
FAh		X		K3, K5 keyboards (Canada)
FBh		X		K2 keyboard (France)
FCh		X		K2 keyboard (Finland/Sweden)
FDh		X		K2 keyboard (Canada)
FEh		X		I-Bus synchronization byte. Do not use.
FFh		X		K2 keyboard (USA)

H

Byte Stream Escape Sequences

Introduction to Appendix H

The following tables describe byte stream escape sequences.

Table H-1. Escape Sequence Redirecting a Video Byte Stream

Sequence	Effect
<hr/> OFFh, 'X'/,	Redirect this video byte stream to frame <i>i</i> where <i>i</i> is a single byte containing (in binary) the number of the required frame.

Table H-2. Escape Sequence Controlling Character Attributes

(Page 1 of 2)

NOTE: When using standard VGA on EISA/ISA-Bus workstations, some attributes (specifically, bold, underlining, struckthrough, and blinking) may display differently than expected. System configuration options allow you to substitute color for these attributes. See the CTOS System Administration Guide for more information.

Sequence*	Effect			
0FFh, 'A', <i>mode</i>	Set the attribute for subsequent characters sent to the frame according to <i>mode</i> , where <i>mode</i> is a single ASCII character defined as follows:			
Mode	Blink	Reverse	Underline	Half-bright
A	no	no	no	no
B	no	no	no	yes
C	no	no	yes	no
D	no	no	yes	yes
E	no	yes	no	no
F	no	yes	no	yes
G	no	yes	yes	no
H	no	yes	yes	yes
I	yes	no	no	no
J	yes	no	no	yes
K	yes	no	yes	no
L	yes	no	yes	yes
M	yes	yes	no	no
N	yes	yes	no	yes
O	yes	yes	yes	no
P	yes	yes	yes	yes

*Each of the 3-byte sequences begins with the escape byte 0FFh and continues with a pair of characters represented by the specified 8-bit ASCII character codes.

Table H-2. Escape Sequence Controlling Character Attributes

(Page 2 of 2)

Sequence*	Effect																					
0FFh, 'A', 'Z'	Enable a mode where writing a character to a character position does not change the character attributes of that character position.																					
0FFh, 'B', <i>mask</i>	Designates some combination of attributes, where <i>mask</i> is an 8-bit binary value as follows:																					
	<table><tr><th>Bit</th><th>Hexadecimal Value</th><th>Attribute</th></tr><tr><td>0</td><td>01h</td><td>Half-Bright</td></tr><tr><td>1</td><td>02h</td><td>Underline</td></tr><tr><td>2</td><td>04h</td><td>Reverse Video</td></tr><tr><td>3</td><td>08h</td><td>Blinking</td></tr><tr><td>4</td><td>10h</td><td>Bold</td></tr><tr><td>5</td><td>20h</td><td>Struck-through</td></tr></table>	Bit	Hexadecimal Value	Attribute	0	01h	Half-Bright	1	02h	Underline	2	04h	Reverse Video	3	08h	Blinking	4	10h	Bold	5	20h	Struck-through
Bit	Hexadecimal Value	Attribute																				
0	01h	Half-Bright																				
1	02h	Underline																				
2	04h	Reverse Video																				
3	08h	Blinking																				
4	10h	Bold																				
5	20h	Struck-through																				
	An example is 0FFh, 'B', 05h, which means an attribute of reverse video and half-bright. Note that this is the same as 0FFh, 'A', 'F'.																					
	This mask byte has the same format as the PutFrameAttrs operation.																					

*Each of the 3-byte sequences begins with the escape byte 0FFh and continues with a pair of characters represented by the specified 8-bit ASCII character codes.

Table H-3. Escape Sequence Controlling Pause

Sequence*	Effect
0FFh, 'P', 'N'	Turn on the pause facility.
0FFh, 'P', 'F'	Turn off the pause facility.

*Each of the 3-byte sequences begins with the escape byte 0FFh and continues with a pair of characters represented by the specified 8-bit ASCII character codes.

Table H-4. Escape Sequence Controlling Screen Attributes

NOTE: When using standard VGA on EISA/ISA-Bus workstations, some attributes (specifically, bold, underlining, struckthrough, and blinking) may display differently than expected. System configuration options allow you to substitute color for these attributes. See the CTOS System Administration Guide for more information.

Sequence*	Effect
0FFh, 'H', 'N'	Turn on the screen half-bright attribute.
0FFh, 'H', 'F'	Turn off the screen half-bright attribute.
0FFh, 'R', 'N'	Turn on the screen reverse video attribute
0FFh, 'R', 'F'	Turn off the screen reverse video attribute.

*Each of the 3-byte sequences begins with the escape byte 0FFh and continues with a pair of characters represented by the specified 8-bit ASCII character codes.

Table H-5. Escape Sequence Controlling Scrolling and Cursor Positioning

Sequence	Effect
0FFh, 'C', x, y	Position the cursor at column x of line y where x and y are single bytes containing (in binary) the column and line numbers. A value of 0FFh for x or y specifies, respectively, the last column or line of the frame.
0FFh, 'S', f, l, c, 'D'	Scroll down a portion of the frame. The portion begins at line f(first) and extends down to, but not including, line l(last). It is scrolled down by c lines and the top lines of the scrolled area are filled with the blank character recorded in the bSpace field of the Video Control Block (VCB). f, l, and c are single bytes containing binary numbers. A value of 0FFh for f or l specifies an imaginary line just below the bottom of the frame.
0FFh, 'S', f, l, c, 'U'	Scroll up a portion of the frame.
0FFh, 'V', 'N'	Make the cursor visible.
0FFh, 'V', 'F'	Make the cursor invisible.

Table H-6. Escape Sequence Performing Miscellaneous Functions

(Page 1 of 2)

Sequence	Effect
0FFh, 'L', 'N'	Enable literal mode, with special character interpretation suppressed (except for the escape character 0FFh). (See Table H-1.) For example, in literal mode, the character code 08h displays a visible backspace symbol rather than performing the backspace function.
0FFh, 'L', 'F'	Disable literal mode.
0FFh, 'E', 'L'	Erase to the end of the current line of the frame. That is, set the characters to the blank character recorded in the <i>bSpace</i> field of the Video Control Block and turn off all attributes.
0FFh, 'E', 'F'	Erase to the end of the current frame.
0FFh, 'F', <i>char</i> , <i>x</i> , <i>y</i> , <i>w</i> , <i>h</i>	Fill an entire rectangle of the current frame with a single character given by the single byte <i>char</i> . The rectangle is specified by four 1-byte binary numbers: the column and line of the upper left corner (<i>x</i> and <i>y</i>), and the width and height (<i>w</i> and <i>h</i>) of the rectangle. A value of 0FFh for <i>x</i> or <i>y</i> specifies, respectively, the last column or line of the frame. A value of 0FFh for <i>w</i> or <i>h</i> specifies, respectively, the remaining width or height of the frame.

Table H-6. Escape Sequence Performing Miscellaneous Functions
(Page 2 of 2)

Sequence	Effect																
0FFh, 'I', <i>led</i> , 'N'	Turn on an LED indicator on the keyboard according to the following table:																
	<table> <tr> <th>LED</th><th>Key</th></tr> <tr> <td>1</td><td>f1</td></tr> <tr> <td>2</td><td>f2</td></tr> <tr> <td>3</td><td>f3</td></tr> <tr> <td>8</td><td>f8</td></tr> <tr> <td>9</td><td>f9</td></tr> <tr> <td>0</td><td>f10</td></tr> <tr> <td>T</td><td>OVER TYPE</td></tr> </table>	LED	Key	1	f1	2	f2	3	f3	8	f8	9	f9	0	f10	T	OVER TYPE
LED	Key																
1	f1																
2	f2																
3	f3																
8	f8																
9	f9																
0	f10																
T	OVER TYPE																
0FFh, 'I', <i>led</i> , 'F'	Turn off an LED indicator on the keyboard according to the table immediately above.																
0FFh, 0FFh	Display a single cross-hatch bar-chart character. The cross-hatch bar-chart character has an 8-bit representation of 0FFh (255) and thus cannot be displayed in any other way.																

Table H-7. Video Byte Stream Interpretation of Special Characters

Hexa- decimal Value	Video Byte Stream Key	Interpretation*
01h	Up arrow	Move the cursor up one line. If the cursor is in the top line of the frame, reposition it to the bottom line.
07h	Cancel	Activate audio alarm for one-half second.
08h	Backspace	Backspace one character (with wraparound) and blank that character.
09h	Tab	Tab to next multiple of eight columns. For example, if the cursor is in columns 0-7, it moves to column 8; if it is in columns 8-15, it moves to column 16.
0Ah	Return	Move to first column of next line; scroll if necessary.
0Bh	Down arrow	Move the cursor down one line. If the cursor is in the bottom line of the frame, reposition it to the top line.
0Ch	Next Page	Blank the frame and position the cursor in its upper left corner.
0Dh	Bound	Ignored.
0Eh	Left arrow	Move the cursor left one character position. If the cursor is in the first column of the frame, reposition it to the last column.
12h	Right arrow	Move the cursor right one character position. If the cursor is in the last column of the frame, reposition it to the first column.
0FFh	Code-Delete	Begin multibyte escape sequence.

*A multibyte escape sequence is available to disable all these special interpretations except 0FFh. (See Table H-6.)

**Table H-8. Submit File Escape Sequences:
Permitted Codes**

Character	Code	Function
¢	03h	Two-character escape sequence that represents the character code 03h. Since 03h (¢) is used to introduce escape sequences, this escape sequence (that is, two consecutive ¢ characters) is the only way to represent the ¢ in a submit file.
1	31h	Three-character read-direct escape sequence. (For details, see "Keyboard and I-Bus Management" in the <i>CTOS Operating System Concepts Manual</i> .)
2	32h	End-of-file escape sequence. When this two-character escape sequence is read during a ReadKbd operation, the submit file is closed. The current and subsequent ReadKbd operations read characters directly from the keyboard. (This escape sequence is meaningful only in submit files that were created as recording files rather than through the Editor.)

System-Common Procedure Parameter Syntax

The character string syntax for defining system-common procedure parameters is shown below. This syntax is in Backus-Naur Form (BNF). Parameters must be defined using this syntax in calls to the SystemCommonInstall operation, which dynamically installs system-common procedures.

<definition string> ::= <definition string><term>

<term> ::= <word term>

| <selector term>

| <pointer term>

| <pointer term><count term>

<word term> ::= w

<selector term> ::= s

<pointer term> ::= <input pointer term>

| <returned pointer term>

<input pointer term> ::= p

<returned pointer term> ::= q

<count term> ::= c

For example, for the call

SampleCall(pBuf, cbBuff, saDataSeg, iReadMax, pcbWrittenRet)

the string defining the parameter list would be

pcswq

where the calling sequence is "pointer, count, selector, word, returned pointer."

The `SystemCommonInstall` operation accepts this character string syntax for protected mode (GDT-based) system-common procedures it installs. (See the *pbParamDef/cbParamDef* parameters to `SystemCommonInstall` in Chapter 3, "Operations.") The operating system can subsequently use this string for creating alias pointers. For example, when the operating system intercepts (traps) calls made by real mode programs to the installed procedure, it creates alias GDT selectors for each pointer it identifies in the character string.

The tables in this appendix group all operations by operation type (Kernel primitive, system-common procedure, request, or library routine). The tables show the internal version number and other pertinent information for each operation.

You can use this appendix to determine whether your system supports an operation. For example, to see if a system operation (Kernel primitive, request, or system-common procedure) is supported, compare the internal version number listed in this appendix with your operating system's version number, which you can obtain by calling `CurrentOsVersion`. To achieve the greatest flexibility in creating portable programs, you should always link your programs with the appropriate version of Standard Software. (See your release documentation for details on the version you should use.)

Table J–1. Kernel Primitives

Kernel Primitive	Kernel Number	Internal Version Number
ChangePriority	10	9.1
ChangeProcessPriority	15	9.1
Check	2	9.1
ControlInterrupt	23	12.0
CreateProcess	3	9.1
DeviceInService	24	12.0
ForwardRequest	19	9.1
KillProcess	17	10.0
MediateIntHandler	9	9.1
NewProcess	16	10.0
PSend	6	9.1
QueryProcessInfo	31	12.3
Request	4	9.1
RequestDirect	11	9.1
RequestRemote	12	9.1
		(SRP only)
RescheduleProcess	18	10.0
ResetTimerInt	8	9.1
Respond	5	9.1
Send	0	9.1
SetDeltaPriority	14	9.1
SetDispMsw287	22	9.6
SetTimerInt	7	9.1
SuspendProcess	27	12.0
SuspendUser	25	12.0
UnsuspendProcess	28	12.0
UnsuspendUser	26	12.0
Wait	1	9.1
WaitLong	21	10.0

Table J-2. Requests

(Page 1 of 14)

Request	Request Code	Internal Version Number
AccessVersion	821	9.13
AddQueue	32865	Queue Manager
AddQueueEntry	137	Queue Manager
AllocAllMemorySL	46	9.1
AllocAreaSL	278	9.1
AllocExch	40	9.1
AllocLdtSlot	333	11.0
AllocMemoryFramesSL	325	11.0
AllocMemoryLL	44	9.1
AllocMemoryPermanent	338	11.0
AllocMemorySL	42	9.1
AllocUserNumbers	308	12.0
AsGetVolume	33171	Audio Service
AsSetVolume	33174	Audio Service
Beep	52	9.1
CdAbortUser	33139	CD-ROM Service
CdAbsoluteRead	33203	CD-ROM Service
CdAudioCtl	33205	CD-ROM Service
CdChangeUserNumber	33142	CD-ROM Service
CdControl	33206	CD-ROM Service
CdClose	33213	CD-ROM Service
CdDirectoryList	33217	CD-ROM Service

Table J-2. Requests

(Page 2 of 14)

Request	Request Code	Internal Version Number
CdGetDirEntry	33207	CD-ROM Service
CdGetVolumeInfo	33208	CD-ROM Service
CdOpen	33212	CD-ROM Service
CdRead	33209	CD-ROM Service
CdSearchClose	33214	CD-ROM Service
CdSearchFirst	33215	CD-ROM Service
CdSearchNext	33216	CD-ROM Service
CdServiceControl	33145	CD-ROM Service
CdSwapUser	33141	CD-ROM Service
CdTerminateUser	33140	CD-ROM Service
CdVerifyPath	33218	CD-ROM Service
CdVersion	33210	CD-ROM Service
CfaFfVersion	8226	12.0
CfamVersion	8224	12.0
CfaWaVersion	8225	12.0
Chain	28	9.1
ChangeCommLineBaudRate	4104	9.4
ChangeFileLength	13	9.1
ChangeOpenMode	215	9.1
CheckpointSysIn	68	9.1
CleanQueue	32866	Queue Manager
ClearPath	02	9.1

Table J-2. Requests

(Page 3 of 14)

Request	Request Code	Internal Version Number
CloseAllFiles	19	9.1
CloseAllFilesLL	62	9.1
CloseFile	10	9.1
ClosePSSession	8197	Performance Statistics
CloseRtClock	50	9.1
CloseTerminal	12307	9.1
		(SRP only)
ConfigureSpooler	188	Spooler
ConfigurationQuery	392	12.0
ConvertToSys	98	9.1
CfaServerVersion	8224	12.0
CfaWaVersion	8225	12.0
CfaServerVersion	8224	12.0
CreateAlias	284	9.1
CreateBigPartition	331	11.0
CreateDir	17	9.1
CreateFile	05	9.1
CreatePartition	181	9.1
CreateUser	296	10.0
DCXVersion	8227	12.0
DeactivateRunFile	305	10.0
DeallocAliasForServer	337	11.0
DeallocExch	41	9.1
DeallocUserNumbers	309	12.0
DeallocMemoryLL	45	9.1
DeallocMemorySL	43	9.1
DeallocRunFile	351	11.2
DeallocSg	345	11.2
DefineLocalPageMap	320	11.0
DeinstPSServer	8198	Performance Statistics
DeinstallMcr	8235	12.1

Table J-2. Requests

(Page 4 of 14)

Request	Request Code	Internal Version Number
DeinstallQueueManager	32867	Queue Manager
DeleteDir	18	9.1
DeleteFile	06	9.1
DisableActionFinish	67	9.1
DisableCluster	122	9.1
		(WS only)
DismountVolume	12	9.1
DmaMapBuffer	436	12.3
DoDirectRead	318	10.0
DoDirectWrite	319	10.0
DrainTerminalOutput	12313	9.1
		(SRP only)
EstablishQueueServer	146	Queue Manager
ExpandAreaLL	280	9.1
ExpandAreaSL	279	9.1
FilterDebugInterrupts	335	11.0
Format	38	9.1
FreeLargeLL	292	10.0
FsCableDrop	334	11.0
GetClusterStatus	100	9.1
GetDateTime	14	9.1
GetDirStatus	216	9.1
GetFhLongevity	31	9.1
GetFileStatus	08	9.1
GetHandleStatus	297	11.0 WS
		10.4 SRP
GetKbdId	65524 or-12	9.1
GetKeyboardId	449	9.13
GetMemoryInfo	306	11.0
GetNodeName	302	3.2 CT-Net
GetPartitionExchange	184	9.1
GetPartitionHandle	177	9.1
GetPartitionStatus	185	9.1

Table J-2. Requests

(Page 5 of 14)

Request	Request Code	Internal Version Number
GetPartitionSwapMode	364	12.0
GetRouteTable	312	12.0
GetQMStatus	32868	(SRP only) Queue Manager
GetScsilInfo	353	11.2
GetUcb	27	9.1
GetUserStatus	272	9.1
GetVhb	15	(WS only) 9.1
GetWsUserName	202	9.1
GoDebugger	4121	11.1
InitCharMap	76	9.1
InitCommLine	4102	(WS only) 9.4
InitLocalPageMap	326	11.0
InitVidFrame	75	9.1
InstallNet	344	(WS only) 10.4
LinkFile	327	(SRP only) 11.0
LoadFile	293	10.0
LoadFontRam	22	9.1
LoadInteractiveTask	262	(WS only except AWS) 9.1
LoadPrimaryTask	178	9.1
LoadRunFile	350	11.2
LoadTask	29	9.1
LocalConfigureSpooler	12356	Spooler (SRP only)

Table J-2. Requests

(Page 6 of 14)

Request	Request Code	Internal Version Number
LocalOpenTape	12352	Spooler (SRP only)
LocalSpoolerPassword	12357	Spooler (SRP only)
LockFile	395	11.3
LockInCache	373	12.0
LockXbis	32797	Xbif Service
MakeDirectory	328	11.0
MapPhysicalAddress	467	12.3
MapXBusWindow*	264	9.1 (WS only)
MapXBusWindowLarge*	321	11.0 (WS only)
MarkKeyedQueueEntry	142	Queue Manager
MarkNextQueueEntry	141	Queue Manager
McrDeinstall	8235	12.1
McrFillBuf	4193	12.1
McrVersion	8228	12.1
MegaframeDisableCluster	12360	9.1 (SRP only)
MountVolume	11	9.1
MouseVersion	8212	12.0
OpenFile	04	9.1
OpenFileLL	97	9.1
OpenPSLogSession	8199	Performance Statistics
OpenPSStatSession	8194	Performance Statistics
OpenRtClock	49	9.1
OpenTerminal	12306	9.1 (SRP only)

*This Request is not supported on shared resource processors.

Table J-2. Requests

(Page 7 of 14)

Request	Request Code	Internal Version Number
OsVersion	261	9.1
PostKbdTable	400	9.13
PSCloseSession	8197	Performance Statistics
PSDeinstServer	24996	Performance Statistics
PSGetCounters	8195	Performance Statistics
PSOpenLogSession	8199	Performance Statistics
PSOpenStatSession	8194	Performance Statistics
PSReadLog	8200	Performance Statistics
PSResetCounters	8196	Performance Statistics
PurgeMcr	8236	12.1
QueueMgrVersion	8210	12.0
QueryBigMemAvail	283	9.1
QueryBoardInfo	314	12.0
QueryDcb	124	9.1
QueryDeviceName	346	11.2
QueryDeviceNames	340	12.0 WS 10.4 SRP
QueryDiskGeometry	366	12.0
QueryExitRunFile	187	9.1
QueryFrameAttrs	113	9.7
QueryFrameString	114	9.7
QueryIoOwner	466	12.3
QueryKbdLeds	55	9.1
QueryKbdState	58	9.1
QueryMemAvail	48	9.1
QueryNodeName	16393	3.2 CT-Net
QueryRequestInfo	271	9.1
QueryTrapHandler	470	12.3
QueryVidHdw	21	9.1

Table J-2. Requests

(Page 8 of 14)

Request	Request Code	Internal Version Number
QueryVidHdw	21	9.1
QueryWsNum	61	9.1
QueueMgrVersion	8210	Queue Manager
Read	35	9.1
ReadActionCode	60	9.1
ReadActionKbd	256	9.1
ReadDirSector	25	9.1
ReadHardId	8192	11.0
ReadKbd	53	9.1
ReadKbdDataDirect	347	11.2
ReadKbdDirect	54	9.1
ReadKbdInfo	403	9.13
ReadKbdSemiCoded	49234	11.2
ReadKbdStatus	257	9.1
ReadKeyedQueueEntry	140	Queue Manager
ReadMcr	8234	12.1
ReadNextQueueEntry	139	Queue Manager
ReadOsKbdTable	399	9.13
ReadPSLog	8200	Performance Statistics
ReadRemote	12325	9.1 (SRP only)
ReadSwap	303	10.0
ReadTerminal	12309	9.1 (SRP only)
RemakeAliasForServer	336	11.0 (IU only)
RemakeFh	4110	10.0
RemoteBoot	12290	9.1 (SRP only)
RemoteBootCopy	311	12.0 (SRP only)

Table J-2. Requests

(Page 9 of 14)

Request	Request Code	Internal Version Number
RemoveDirectory	329	11.0
RemoveKeyedQueueEntry	138	Queue Manager
RemoveMarkedQueueEntry	143	Queue Manager
RemovePartition	176	9.1
RemoveQueue	32869	Queue Manager
RenameFile	07	9.1
ReopenFile	294	10.0
RescheduleMarkedQueueEntry	33119	Queue Manager
ReserveMemoryPartition	354	11.2
ResetAllSegs	291	10.0
ResetCommLine	4103	10.0
ResetDeviceHandler	469	12.3
ResetPSCounters	8196	Performance Statistics
ResetTrapHandler	461	12.3
ResetVideoGraphics	33137	12.0
ResetIbusHandler	349	11.2
ResetMemoryLL	47	9.1
ResetMemorySL	290	10.0
ResetVideo	74	9.1
ResetVideoGraphics	33137	12.0
ResetXBusMIsr	32800	Xbif Service
RewriteMarkedQueueEntry	145	Queue Manager
RkvsVersion	8208	12.0
ScreenPrintVersion	8229	9.13
ScsiCdbDataIn	360	12.0
ScsiCdbDataOut	361	12.0
ScsiClosePath	358	12.0
ScsiManagerNameQuery	363	12.0

Table J–2. Requests

(Page 10 of 14)

Request	Request Code	Internal Version Number
ScsiOpenPath	355	12.0
ScsiQueryPathParameters	356	12.0
ScsiQueryInfo	353	12.0
ScsiRequestSense	362	12.0
ScsiReset	359	12.0
ScsiSetPathParameters	357	12.0
ScsiTargetCdbCheck	389	12.0
ScsiTargetCdbWait	390	12.0
ScsiTargetDataReceive	387	12.0
ScsiTargetDataTransmit	388	12.0
ScsiTargetOperationsAbort	405	12.0
SeqAccessCheckpoint	33234	Sequential Access Service
SeqAccessClose	33227	Sequential Access Service
SeqAccessControl	33230	Sequential Access Service
SeqAccessDiscardBufferData	33236	Sequential Access Service
SeqAccessModeQuery	33228	Sequential Access Service
SeqAccessModeSet	33229	Sequential Access Service
SeqAccessOpen	33226	Sequential Access Service
SeqAccessRead	33233	Sequential Access Service
SeqAccessRecoverBufferData	33235	Sequential Access Service
SeqAccessStatus	33231	Sequential Access Service
SeqAccessTermination	36932	Sequential Access Service
SeqAccessVersion	33225	Sequential Access Service

Table J-2. Requests

(Page 11 of 14)

Request	Request Code	Internal Version Number
SeqAccessWrite	33232	Sequential Access Service
SerialNumberQuery	391	12.0
ServeRq	99	9.1
ServiceOverlayA	323	11.0
ServiceOverlayB	324	11.0
Set386TrapHandler	322	11.0
SetColorData	286	9.1
SetDateTime	51	9.1
SetDefaultTrapHandler	339	11.0
SetDeviceHandler	310	12.0
SetDevParams	16	9.1
SetDirStatus	217	9.1
SetDiskGeometry	367	12.0
SetExitRunFile	186	9.1
SetFhLongevity	30	9.1
SetFileStatus	09	9.1
SetFSConfigParams	352	11.2
SetIBusHandler	348	11.2
SetIntHandler	69	9.1
SetIoOwner	464	12.3
SetKbdLed	56	9.1
SetKbdSemicodedMode	49232	11.2
SetKbdUnencodedMode	57	9.1
SetKeyboardOptions	448	9.13
SetLPlsr	121	9.1
SetNode	252	9.1
SetPartitionExchange	183	9.1
SetPartitionLock	182	9.1
SetPartitionSwapMode	365	12.0
SetPath	01	9.1
SetPrefix	03	9.1
SetScreenVidAttr	77	9.1
SetSegmentAccess	285	9.1
SetSysInMode	59	9.1

Table J-2. Requests

(Page 12 of 14)

Request	Request Code	Internal Version Number
SetTerminal	12310	9.1 (SRP only)
SetTrapHandler	268	9.1
SetVideoTimeout	255	9.1
SetWsUserName	203	9.1
SetXBusMIsr	32799	Xbif Service
SgFromSa	332	11.0
ShrinkAreaLL	282	9.1
ShrinkAreaSL	281	9.1
ShrinkPartition	341	11.2
SignOnLog	4155	11.2
SpoolerPassword	189	Spooler
SpoolerVersion	8211	12.0
StatisticsVersion	8230	12.0
SwapInContext	295	10.0
SwapInContextAndLock	4112	12.0
TerminateMcr	65520 or -16	9.1
TerminatePartitionTasks	179	9.1
TerminateQueueServer	147	Queue Manager
TerminateVideoClient	4118	12.3
TsConnect	32814	Voice/ Data
TsDataChangeParams	32810	Voice/ Data
TsDataCheckpoint	32808	Voice/ Data
TsDataCloseLine	32809	Voice/ Data
TsDataOpenLine	32805	Voice/ Data
TsDataRead	32806	Voice/ Data

Table J-2. Requests

(Page 13 of 14)

Request	Request Code	Internal Version Number
TsDataRetrieveParams	32811	Voice/ Data
TsDataUnacceptCall	32812	Voice/ Data
TsDataWrite	32807	Voice/ Data
TsDeinstall	32824	Voice/ Data
TsDial	32815	Voice/ Data
TsDoFunction	32817	Voice/ Data
TsGetStats	32813	Voice/ Data
TsHold	32820	Voice/ Data
TsLoadCallProgressTones	33201	Voice/ Data
TsOffHook	32818	Voice/ Data
TsOnHook	32819	Voice/ Data
TsQueryConfigParams	32821	Voice/ Data
TsReadTouchTone	32816	Voice/ Data
TsRing	32823	Voice/ Data
TsSetConfigParams	32822	Voice/ Data
TsVersion	8216	12.0
TsVoiceConnect	32803	Voice/ Data
TsVoicePlaybackFromFile	32801	Voice/ Data

Table J-2. Requests

(Page 14 of 14)

Request	Request Code	Internal Version Number
TsVoiceRecordToFile	32802	Voice/ Data
TsVoiceStop	32804	Voice/ Data
UnlockFile	396	11.3
UnlockInCache	374	12.0
UnlockXbis	32798	Xbif Service
UnmapPhysicalAddress	468	12.3
UnmarkQueueEntry	144	Queue Manager
UpdateDcb	12324	9.1 (SRP only)
UpdateFPMountTable	12304	9.1 (SRP only)
UpdateRouteTable	313	12.0 (SRP only)
VacatePartition	180	9.1
WhereTerminalBuffer	12308	9.1 (SRP only)
Write	36	9.1
WriteHardId	8193	11.3
WriteKbdBuffer	402	9.13
WriteLog	125	9.1
WriteRemote	12326	9.1 (SRP only)
WriteSwap	304	10.0
XbifVersion	8217	12.0

Table J-3. System-Common Procedures

(Page 1 of 5)

System-Common Procedure	Procedure Number	Internal Version Number
AssignKbd	8	9.1
AssignVidOwner	15	9.1
BuildLdtSlot	125	11.0
CacheClose	153	12.0
CacheFlush	152	12.0
CacheGetEntry	149	12.0
CacheGetStatistics	163	12.0
CacheGetStatus	151	12.0
CacheInit	148	12.0
CacheReleaseEntry	150	12.0
CallRealCommlsr	133	11.0
CheckpointBsLP	29	9.1
CheckUserActive	170	9.11
		(BTOS real mode only)
		11.07
		(BTOS protected mode only)
CodeIKbd	200	9.13
Crash	16	9.1
DbgBmPutChars	120	11.0
DbgBmScroll	121	11.0
DbgBmSwitch	122	11.0
DbgPosCursor	147	12.0
DbgPutChars	120	12.0
DbgScroll	121	12.0
DefineInterLevelStack	102	11.0
Delay	14	9.1
DirtyCode	161	12.0
DmaMapBufferFast	203	12.3
DmaUnmapBuffer	204	12.3
Doze	169	11.07
		(BTOS protected mode only)
		12.0
EnterBootRom	131	11.0
ErrorExit	10	9.1
ErrorExitUser	186	12.3

Table J-3. System-Common Procedures

(Page 2 of 5)

System-Common Procedure	Procedure Number	Internal Version Number
Exit	11	9.1
ExitAndRemove	75	10.0
		(WS only)
ExtractRunFileHeader	148	12.3
FillBufferLP	140	11.2
FillFrame	23	9.1
FillFrameRectangle	24	10.0
FlushBufferLP	28	9.1
FrameBackSpace	26	9.1
FsCanon	107	11.0
FSrpUp	128	11.0
GetClusterId	2043	9.11
		(BTOS real mode only)
		11.07
		(BTOS protected mode only)
GetCommLineDmaStatus	141	11.2
GetDaiNumber	136	11.1
GetFRmosUser	104	11.0
GetLocalDaiNumber	144	11.2
GetModuleAddress	216	12.3
GetModuleId	44	9.1
GetPAscb	31	9.1
GetProcInfo	45	9.1
		(SRP only)
GetPStructure	47	9.1
GetSlotInfo	46	9.1
		(SRP only)
GetSlotFromName	162	12.0
		(SRP only)
GetUserNumber	30	9.1
gfx_BuildRasterText	97	12.0
gfx_ChangeCursor	91	12.0
gfx_DefineCursor	85	12.0
gfx_Draw	95	12.0
gfx_GetContiguousBm	96	12.0
gfx_HandleCursorIrpt	90	12.0

Table J-3. System-Common Procedures

(Page 3 of 5)

System-Common Procedure	Procedure Number	Internal Version Number
gfx_LockRasterRegion	80	12.0
gfx_PosCursor	87	12.0
gfx_QueryCursorPosition	94	12.0
gfx_RasterOp	82	12.0
gfx_RasterOpText	98	12.0
gfx_RemoveCursor	88	12.0
gfx_ScrollRasterLines	84	12.0
gfx_SetCursor	86	12.0
gfx_SetCursorBlinking	89	12.0
gfx_SetCursorColor	93	12.0
gfx_SetCursorPriority	92	12.0
gfx_UnlockRaster	81	12.0
LockInContext	159	12.0
LockVideo	48	9.1
LockVideoForModify	50	9.1
MapBusAddress	155	12.0
MapSgUserNum	127	11.0
MoveFrameRectangle	73	10.0
NotifyVidMemLineUser	124	11.0
OpenByteStreamLP	27	9.1
PaFromSn	101	11.0
PosFrameCursor	17	9.1
ProcessKeys	198	9.13
PutFrameAttrs	18	9.1
PutFrameChars	19	9.1
PutFrameCharsAndAttrs	69	10.0
QueryCoproprocessor	138	11.0
QueryDefaultRespExch	0	9.1
QueryFrameChar	20	9.1
QueryFrameCharsAndAttrs	70	10.0
QueryFrameCursor	72	10.0
QueryFrameBounds	71	10.0
QueryLdtR	132	11.0
QueryLoadAddress	77	12.0

Table J–3. System-Common Procedures

(Page 4 of 5)

System-Common Procedure	Procedure Number	Internal Version Number
QueryModel	76	12.0
QueryProcessNumber	1	9.1
QueryUserLocation	157	12.0
ReceiveCommLineDma	142	11.2
ReadCommLineStatus	56	10.0
ReleaseByteStreamLP	43	9.1
ResetFrame	21	9.1
ResizeloMap	215	12.3
ReuseAlias	103	11.0
ReuseAliasLarge	126	11.0
ScrollFrame	22	9.1
ServiceOverlayC	105	11.0
SetEnvironment	149	12.3
SetLdtRDs	134	11.0
SetPStructure	78	10.0
SetSwapDisable	9	9.1
SetUpOKeys	197	9.13
SnFromSr	109	11.0
SrFromSn	100	11.0
StringsEqual	5	9.1
SwapDebuggerVideo	146	12.0
SwapXBusEar	52	9.1
SystemCommonCheck	262	11.0
SystemCommonConnect	256	11.0
SystemCommonInstall	129	11.0
SystemCommonQuery	130	11.0
TransmitCommLineDma	143	11.2
ULCmpB	3	9.1
UnlockInContext	160	12.0
UnlockVideo	49	9.1
UnlockVideoForModify	51	9.1
UnmapBusAddress	156	12.0
UpdateFSDeviceName	158	12.0

Table J-3. System-Common Procedures

(Page 5 of 5)

System-Common Procedure	Procedure Number	Internal Version Number
UpdateChordState	221	9.13
VamStatus	79	12.0
WriteCommLineStatus	56	10.0
WriteIBusDevice	145	11.2
WriteIBusEvent	139	11.0
		(WS only)
XlateChar	196	9.13

Table J-4. Standard Operating System Library Procedures

(Page 1 of 9)

System-Library Procedure	Internal Version Number
AcquireByteStream	11.0
AltWildCardNext	12.1
AtFileInit	11.3
AtFileNext	11.3
BuildFileSpec	12.0
BuildSpecFromNode	12.0
BuildSpecFromVol	12.0
BuildSpecFromDir	12.0
BuildSpecFromFile	12.0
BuildSpecFromPassword	12.0
BuildFullSpecFromPartial	12.0
CheckErc	11.0
CheckForOperatorRestartC	11.0
CheckpointBs	11.0
CheckpointBsAsyncC	11.0

Table J-4. Standard Operating System Library Procedures

(Page 2 of 9)

System- Library Procedure	Internal Version Number
CheckpointBsC	11.0
CheckpointRsFile	11.0
CheckReadAsync	11.0
CheckWriteAsync	11.0
CloseAltMsgFile	12.0
CloseByteStream	11.0
CloseDaFile	11.0
CloseErcFile	12.0
CloseMsgFile	11.0
CloseRsFile	11.0
CloseServerMsgFile	11.0
CloseSysCmds	12.0
CloseVidFilter	12.0
CompactDateTime	11.0
ConfigCloseFile	12.1
ConfigGetNextToken	12.1
ConfigGetRestOfLine	12.1
ConfigOpenFile	12.1
ConfigSetPosition	12.1
CopyFile	12.0
CParams	11.0
CreateExecScreen	12.0
CSubParams	11.0
CurrentOsVersion	11.0
DeallocateRods	11.0
DeallocVlpb	12.0
DeleteByteStream	12.0
DeleteDaRecord	11.0
DiscardInputBsC	11.0
DiscardOutputBsC	11.0
EnableSwapperOptions	11.0
EnIsCase	12.1
EnIsClass	12.1
EnIsCbToCCols	12.1

Table J-4. Standard Operating System Library Procedures

(Page 3 of 9)

System- Library Procedure	Internal Version Number
EnIsDeleteChar	12.1
EnIsDrawBox	12.1
EnIsDrawFormChars	12.1
EnIsDrawLine	12.1
EnIsAppendChar	12.1
EnIsFieldEdit	12.1
EnIsFieldEditByChar	12.1
EnIsFindC	12.1
EnIsFindRC	12.1
EnIsGetChar	12.1
EnIsGetCharWidth	12.1
EnIsGetPrevChar	12.1
EnIsInsertChar	12.1
EnIsMapCharToStdValue	12.1
EnIsMapStdValueToChar	12.1
EnIsQueryBoxSize	12.1
ErrorExitString	11.0
ExpandDateTime	11.0
ExpandLocalMsg	11.0
FatalError	11.0
FComparePointer	11.0
FieldEdit	12.0
FillBufferAsyncC	11.0
FillBufferC	11.0
FlushBufferAsyncC	11.0
FlushBufferC	11.0
FMergedOs	12.1
FormatDateTime	11.0
FormatTime	11.0
FormatTimeDt	11.0
FormatTimeTm	11.0
FormEdit	12.0
FProcessorSupportsProtectedMode	11.0
FProtectedMode	11.0

Table J-4. Standard Operating System Library Procedures

(Page 4 of 9)

System- Library Procedure	Internal Version Number
FRmos	11.0
FValidPbCb	12.0
GenResString	12.0
GetAltMsg	12.0
GetAltMsgUnexpanded	12.0
GetAltMsgUnexpandedLength	12.0
GetCanonicalNodeAndVol	12.0
GetCtosDiskPartition	12.1
GetBoardInfo	12.0
GetBsLfa	11.0
GetCoproprocessorStatus	11.0
GetCParasOvlyZone	11.0
GetDirInfo	12.0
GetErc	12.0
GetErcLength	12.0
GetFileErc	12.0
GetMsg	11.0
GetMsgUnexpanded	11.0
GetMsgUnexpandedLength	12.0
GetNlsDateName	11.0
GetNlsDateTimeTemplate	12.1
GetNlsKeycapText	11.0
GetNlsTable	12.0
GetOvlyStats	11.0
GetPNlsTable	11.0
GetRsLfa	11.0
GetSegmentLength	12.0
GetServerMsg	11.0
GetSysCmdInfo	12.0
GetStamFileHeader	11.0
GetStandardErcMsg	11.2
GetUserFileEntry	12.0

Table J-4. Standard Operating System Library Procedures
(Page 5 of 9)

System- Library Procedure	Internal Version Number
GetUserLocation	12.0
InitAltMsgFile	12.0
InitErcFile	12.0
InitLargeOverlays	11.0
InitMsgFile	11.0
InitOverlays	11.0
InitSysCmds	12.0
KeyboardProfile	12.1
LoadBackgroundPalette	12.0
LoadColorStyleRam	11.0
LockIn	11.0
LockOut	11.0
LookUpField	11.0
LookUpNumber	11.0
LookUpReset	11.0
LookUpString	11.0
MakePermanent	11.0
MakePermanentP	11.0
MakeRecentlyUsed	11.0
MapCslOvly	11.0
MapIOvlyCs	11.0
MapPStubPProc	11.0
MenuEdit	12.0
Mode3DmaReload	11.0
MoveOverlays	11.0
McrVersion	12.1
NlsCase	11.0
NlsClass	11.0
NlsCollate	11.0
NlsFormatDateTime	11.0
NlsGetYesNoStrings	12.0
NlsGetYesNoStringSize	12.0
NlsNumberAndCurrency	11.0
NlsParseTime	11.0

Table J-4. Standard Operating System Library Procedures

(Page 6 of 9)

System- Library Procedure	Internal Version Number
NIsSpecialCharacters	11.3
NIsStdFormatDateTime	11.0
NIsULCmpB	11.0
NIsVerifySignatures	11.0
NIsYesNoOrBlank	11.0
NIsYesOrNo	11.0
NPrint	11.0
ObtainAccessInfo	12.1
ObtainUserAccessInfo	12.1
OpenByteStream	11.0
OpenByteStreamC	11.0
OpenDaFile	11.0
OpenNlsFile	12.0
OpenRsFile	11.0
OpenServerMsgFile	11.0
OpenUserFile	12.0
OpenVidFilter	12.0
OutputBytesWithWrap	11.0
OutputQuad	11.0
OutputToVid0	11.0
OutputWord	11.0
PaFromP	11.0
ParseFileSpec	12.0
ParseSpecForDir	12.0
ParseSpecForFile	12.0
ParseSpecForNode	12.0
ParseSpecForPassword	12.0
ParseSpecForVol	12.0
ParseTime	11.0
PrintAltMsg	12.0
PrintErc	12.0
PrintFileClose	12.0
PrintFileOpen	12.0
PrintFileStatus	12.0

Table J—4. Standard Operating System Library Procedures

(Page 7 of 9)

System- Library Procedure	Internal Version Number
PrintMsg	11.0
PutCharsAndAttr	12.0
ProgramColorMapper	11.0
PutBackByte	11.0
PutByte	11.0
PutChar	11.0
PutPointer	11.0
PutQuad	11.0
PutWord	11.0
QueryBounds	12.0
QueryCharsAndAttr	12.0
QueryCursor	12.0
QueryDaLastRecord	11.0
QueryDaRecordStatus	11.0
QueryMail	11.0
QueryMemAvail	11.0
QueryModulePosition	11.0
QueryNodeName	12.0
QueryVidBs	11.0
QueryVideo	11.0
QueryZoomBoxPosition	12.0
QueryZoomBoxSize	12.1
ReadAsync	11.0
ReadBsRecord	11.0
ReadByte	11.0
ReadBytes	11.0
ReadByteStreamParameterC	11.0
ReadDaFragment	11.0
ReadDaRecord	11.0
ReadKeySwitch	12.0
ReadRsRecord	11.0
ReadStatusC	11.0
ReadToNextField	11.0
ReinitLargeOverlays	11.0

Table J-4. Standard Operating System Library Procedures

(Page 8 of 9)

System- Library Procedure	Internal Version Number
ReinitOverlays	11.0
ReinitStubs	11.0
ReleaseByteStream	11.0
ReleaseByteStreamC	11.0
ReleasePermanance	11.0
ReleaseRsFile	11.0
RemoveFfsBrckets	12.0
RenameByteStream	12.0
RgParam	11.0
RgParamInit	11.0
RgParamSetEltNext	11.0
RgParamSetListStart	11.0
RgParamSetSimple	11.0
SbPrint	11.0
ScanToGoodRsRecord	11.0
ScrubFile	12.0
ScsiCheckCdbAsync	12.0
ScsiCdbDataInAsync	12.0
ScsiCdbDataOutAsync	12.0
ScsiTargetDataReceiveAsync	12.0
ScsiTargetDataTransmitAsync	12.0
ScsiWaitCdbAsync	12.0
ScsiWaiTargetDataAsync	12.0
SendBreakC	11.0
SerialNumberOldOsQuery	12.1
SetAlphaColorDefault	11.0
SetBsLfa	11.0
SetDaBufferMode	11.0
SetDateTimeMode	12.0
SetField	12.0
SetFieldNumber	12.0
SetImageMode	11.0
SetImageModeC	11.0

Table J-4. Standard Operating System Library Procedures

(Page 9 of 9)

System- Library Procedure	Internal Version Number
SetMsgRet	11.0
SetPartitionName	11.0
SetRsLfa	11.0
SetStyleRam	11.0
SetStyleRamEntry	11.0
SetUserFileEntry	12.0
ShortDelay	12.0
StringsEqual	11.0
TextEdit	11.0
TruncateDaFile	11.0
UndeleteFile	12.0
UnzoomBox	12.0
UpdateOverlayLru	11.0
WildCardClose	12.1
WildCardInit	11.0
WildCardMatch	11.0
WildCardNext	11.0
WriteAsync	11.0
WriteBsRecord	11.0
WriteByte	11.0
WriteByteStreamParameteC	11.0
WriteDaFragment	11.0
WriteDaRecord	11.0
WriteRsRecord	11.0
WriteStatusC	11.0
ZoomBox	12.0
ZPrint	11.0

Table J–5. Mouse Library Procedures

Mouse Library Procedure	Internal Version Number
GetIBusDevInfo	12.0
PDAssignMouse	12.0
PDGetCursorPos	12.0
PDGetCursorPosNSC	12.0
PDInitialize	12.0
PDLoadCursor	12.0
PDLoadSystemCursor	12.0
PDQueryControls	12.0
PDQuerySystemControls	12.0
PDReadCurrentCursor	12.0
PDReadIconFile	12.0
PDSetControls	12.0
PDSetCursorDisplay	12.0
PDSetCursorPos	12.0
PDSetCursorPosNSC	12.0
PDSetCursorType	12.0
PDSetSystemControls	12.0
PDSetTracking	12.0
PDSetCharMapVirtualCoordinates	12.0
PDSetVirtualCoordinates	12.0
PDSetMotionRectangle	12.0
PDSetMotionRectangleNSC	12.0
PDTranslateNSCtoVC	12.0
PDTranslateVCToNSC	12.0
ReadInputEvent	12.0
ReadInputEventNSC	12.0

**Table J-6. Asynchronous System Service
Library Procedures**

Async Library Procedure	Internal Version Number
AllocMemoryInit	12.0
AsyncRequest	12.0
AsyncRequestDirect	12.0
BuildAsyncRequest	12.0
BuildAsyncRequestDirect	12.0
CheckContextStack	12.0
CreateContext	12.0
HeapAlloc	12.0
HeapFree	12.0
HeapInit	12.0
LogMsgIn	12.0
LogRespond	12.0
LogRequest	12.0
ResumeContext	12.0
SwapContextUser	12.0
TerminateAllOtherContexts	12.0
TerminateContext	12.0
TerminateContextUser	12.0

K

Resource Type Codes

Introduction to Appendix K

This appendix lists the predefined resource type codes.

NOTE: Type codes 1 through 15 are MicroSoft codes; type codes 2000 through 2002 are Unisys codes.

Table K-1. Resource Type Codes

(Page 1 of 2)

Resource Type	Type Code	Definition
RT_POINTER	1	Mouse pointer shape
RT_BITMAP	2	Bitmap
RT_MENU	3	Menu template
RT_DIALOG	4	Dialog template
RT_STRING	5	String tables
RT_FONTDIR	6	Font directory
RT_FONT	7	Font
RT_ACCELTABLE	8	Accelerator tables
RT_RCDATA	9	Binary data
RT_MESSAGE	10	Error message tables
RT_DLGINCLUDE	11	Dialog include file name
RT_VKEYTBL	12	Key to vkey tables

Table K-1. Resource Type Codes
(Page 2 of 2)

Resource Type	Type Code	Definition
RT_KEYTBL	13	Key to UGL tables
RT_CHARTBL	14	Glyph to character tables
RT_DISPLAYINFO	15	Screen display information
RT_DEBUGGER	2000	Symbol file
RT_KBDTRANSDATA	2001	Keyboard translation data block
RT_KBDEMULDATA	2002	Keyboard emulation data block



43574714-110